ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE

BUSTRACK

Realizado por
Alberto Castro Olea
Tutorizado por
Enrique Alba Torres
Jamal Toutouh El Alamin
Departamento
LENGUAJE Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Noviembre de 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: Fomentar el uso del transporte público es uno de los pilares de la movilidad inteligente. En el presente Trabajo de Fin de Grado se ha tratado la elaboración de un sistema de recolección de estadísticas útiles sobre el uso de los autobuses públicos (BusTrack). Con esto, se extraerá información útil como por ejemplo la contaminación evitada o las líneas más y menos saturadas. BusTrack consta principalmente de dos partes: una aplicación móvil para Android realizada con Xamarin, que recolecte la información sobre los viajes, y un servidor que contenga una aplicación web, realizada con .NET Core, donde se guardarán y analizarán todos los datos de manera anónima. Para utilizar BusTrack, es necesario crearse una cuenta a través de la aplicación Android, la cual se puede configurar con distintos parámetros para la detección de viajes a través de las redes wifi de los autobuses. Además, se pueden consultar las estadísticas personales y las últimas rutas realizadas en un mapa. Con tal de tener el máximo de seguridad posible, se ha desarrollado un sistema de OAuth2 propio, con el cual el envío de datos sea lo más confidencial posible. Por último, es posible acceder a una aplicación web que muestra la información general en forma de estadísticas.

Palabras claves: autobús, contaminación, aplicación móvil, aplicación web, movilidad inteligente, estadísticas, wifi

Abstract: Promoting the use of public transport is one of the main objectives of smart mobility. In the present final project, we have developed a system to gather information about the use of public buses (BusTrack). Thus, we will be able to extract useful information like avoided pollution or the most and the least used bus lines. BusTrack has consisted mainly of two parts: a mobile application, which gathers the information about travels, and a web application, which stores and analyses the data anonymously. To use BusTrack, it is necessary to create an account through the mobile application, which can be configured by the users with different parameters for detecting bus trips through wifi networks provided by buses. Furthermore, the user can consult his personal stats and his latest routes on a map. In order to have as maximum security as possible, we have developed our own OAuth2 system, which allows the mobile application to send data in a confidential manner. Moreover, it is possible to access the web application to consult general statistics about the bus trips.

Keywords: bus, pollution, mobile application, web application, smart mobility, statistics, wifi

Índice

Capítulo 1
Introducción

Es evidente que la tecnología ha avanzado en múltiples ámbitos de la sociedad, hasta el punto en el cual se integra en la propia ciudad para mejorar la vida de los residentes. Así pues, hoy en día se habla de *Smart City* (Ciudades inteligentes) [1]. En uno de los ámbitos donde se están invirtiendo un mayor esfuerzo es en la aplicación de tecnología para mejorar la movilidad, puesto que afecta de forma drástica a la vida diaria de los ciudadanos.

De este modo, aparece la movilidad inteligente o *smart mobility*. Los principales objetivos de ésta son optimizar los traslados (hacerlos más cortos tanto en distancia como en tiempo), mejorar la seguridad durante el viaje y, además, reducir las emisiones que se generan [2]. Uno de los objetivos claros a la hora de aplicar estas soluciones es el transporte público, puesto que está siendo clave para mejorar la movilidad en las ciudades.

Como parte de esta respuesta, planteamos el sistema de recolección de estadísticas al que llamamos BusTrack. Este utiliza el servicio wifi de los autobuses urbanos para extraer información de los trayectos que hacen los usuarios, para así recolectar estadísticas que puedan ser de utilidad tanto para instituciones como para las personas que usan la aplicación. Un ejemplo de información útil a los usuarios, es el tiempo que pasan en trayectos de bus por la ciudad. En el caso de las instituciones, es la información sobre las líneas más o menos transitadas, así como la contaminación evitada [3].

La detección de que se está realizando un viaje en autobús se realiza mediante el dispositivo móvil. Éste escanea las redes wifi y reconoce la que pertenece al autobús para luego detectar la línea y las paradas del viaje. Esto se ha conseguido mediante la extracción de datos desde la página de la EMT (Empresa Municipal de Transportes de Málaga). Luego de haber realizado el viaje, estos datos son enviados al servidor web para elaborar estadísticas.

1.1. Objetivos

El objetivo del presente Trabajo de Fin de Grado (TFG) es el desarrollo de un sistema informático que recolecte datos estadísticos mediante la aplicación Android sobre el uso de autobuses. A partir de estos, se realizará un análisis sobre los mismos empleando una aplicación web que extrae información de utilidad para los usuarios, como, por ejemplo, el uso que hacen del autobús. Esta recolección se va a completar de forma transparente al usuario empleando su dispositivo móvil e información de la red inalámbrica dentro de los autobuses. Esta información será de utilidad para las instituciones como se ha mencionado anteriormente.

Para cumplir con este objetivo, se ha desarrollado una aplicación Android que detecta cuando un usuario ha subido o bajado de un autobús y guarda la localización del lugar de la detección. Además, se ha desarrollado una aplicación web que almacena los datos de la aplicación Android y realiza las estadísticas, tanto a nivel de usuario como de sistema.

1.2. Estructura de la memoria

La memoria del actual TFG se ha estructurado en capítulos que se cubren las etapas seguidas en cualquier desarrollo de software moderno. A continuación, se introduce el contenido de cada capítulo:

Capítulo 2: Tecnologías y herramientas usadas

En este capítulo se explicarán las herramientas y tecnologías usadas, así como los motivos por los cuales se han escogido estas como las más adecuadas en el desarrollo del proyecto.

Capítulo 3: Especificación y análisis

Aquí se realiza la fase de obtención de los requisitos funcionales y no funcionales de las aplicaciones. Además, por cada requisito funcional, se especificarán los casos de uso más significativos, así como los actores.

Capítulo 4: Diseño

En este capítulo se exponen los diagramas UML de distribución y de clases. Además, se explican los modelos entidad/relación de las bases de datos tanto local como remota, ya que son ligeramente distintas.

Capítulo 5: Implementación y pruebas

Durante la fase del desarrollo, se explicarán las partes más importantes o destacables del proyecto, así como los problemas encontrados y sus soluciones. Además, se comenta brevemente la estructura del proyecto.

Capítulo 6: Conclusiones y posibles líneas futuras

Por último, se realizará un resumen de todo lo aprendido en el proyecto, detallando los puntos más importantes que se han ido encontrando en el desarrollo del mismo. Además, se puntualizan las posibles líneas futuras que puede tener el sistema.

Capítulo 2
Tecnologías y herramientas usadas

En este capítulo se analizarán las tecnologías usadas para la realización del proyecto, así como de una breve explicación del por qué se han usado esas herramientas y no otras.

2.1. Repositorio de código fuente

Los repositorios de código fuente permiten la centralización del proyecto que se está desarrollando en un servidor de alojamiento. Esto permite, en el caso de un equipo, el trabajo simultáneo en un mismo proyecto, haciendo de manera mucho más eficiente el mantenimiento del mismo gracias al control de versiones. En este caso, se ha optado por una de las soluciones más extendidas que es el uso de Git alojado en GitHub. Esto se debe a que es una herramienta muy potente. Más específicamente el repositorio utilizado para el desarrollo del proyecto se encuentra en el siguiente enlace: https://github.com/Infernage/TFG2016

2.2. Aplicación Android

Se ha utilizado Xamarin [4] en el desarrollo la aplicación Android, ya que es una tecnología muy reciente y tiene una gran serie de ventajas, gracias al lenguaje C# [5] y la máquina virtual Mono, con respecto a Android nativo.

2.3. Interfaz web

Para la interfaz web se han usado tres tecnologías distintas. Las principales son HTML y el *framework* Bootstrap [6] que contiene diversas plantillas y funcionalidades para facilitar el desarrollo de interfaces web. Para la lógica en el navegador, se ha utilizado Javascript con las bibliotecas JQuery y FusionCharts [7].

2.4. Servidor web

La aplicación web ha sido realizada con .NET Core [8], la cual trabaja con el lenguaje C#. Esta tecnología trae por defecto dos servidores web, de los cuales se ha usado Kestrel, ya que es el único que es multiplataforma y la gran mayoría de servidores físicos cuentan con un sistema operativo linux.

Se ha utilizado un servidor proxy, debido a que el servidor Kestrel no está creado para desplegarlo directamente a internet. Así, se ha escogido Nginx debido a su gran rendimiento.

2.5. Bases de datos

Para el almacenamiento de datos, se han utilizado dos gestores de bases de datos distintos para la aplicación Android y la aplicación web. En el caso de la aplicación móvil, se ha utilizado la base de datos Realm [9] debido a que su rendimiento y facilidad de manejo es muy superior a la ampliamente usada SQLite. En la aplicación web se ha utilizado la base de datos PostgreSQL [10], ya que ofrece, de manera nativa, varios tipos de datos para localizaciones geográficas.

2.6. Entorno de desarrollo

Por último, para el desarrollo del proyecto se ha utilizado Visual Studio, ya que se usan en todo momento tecnologías basadas en el lenguaje C#. Además, para extraer los datos de las líneas y paradas de la EMT se ha utilizado el *framework* Qt con el entorno de desarrollo QtCreator [11], el cual está basado en C++.

Capítulo 3
Especificación y análisis

En este capítulo, se realiza la fase del análisis de requisitos, tanto funcionales como no funcionales, así como la especificación de los casos de uso más significativos en el sistema y sus actores.

3.1. Requisitos funcionales de la aplicación Android

En esta sección se van a listar los requisitos funcionales que se han ido obteniendo en las reuniones con los tutores. Estos son los que determinan las funcionalidades de BusTrack y lo que pueden realizar los diferentes tipos de actores.

Dado que existe más de un tipo de usuario, se han agrupado los requisitos en dos categorías divididas en tablas: usuario no autenticado (ver Tabla 1) y usuario autenticado (ver Tabla 2 y Tabla 3).

ID	Requisito	Descripción
NAU01	Crear cuenta	El usuario podrá crearse una cuenta en el sistema para poder utilizar todas las funcionalidades que este le ofrece.
NAU02	Autenticarse en el sistema	El usuario tendrá que introducir sus credenciales para poder autenticarse y que el sistema lo reconozca como un usuario con cuenta ya creada.
NAU03	Recuperar cuenta	El usuario podrá recuperar su cuenta en el caso de que haya perdido su contraseña o no se acuerde de la misma, a través de su correo electrónico.

Tabla 1: Requisitos funcionales de usuario no autenticado

ID	Requisito	Descripción
AU01	Añadir red bus	La principal funcionalidad de la aplicación Android, se basa en la detección de viajes bus a través de redes wifi, por lo tanto, se le debe ofrecer al usuario una opción para añadir qué redes deben ser marcadas como redes bus.
AU02	Eliminar red bus	Así como se da la opción de añadir redes bus, deben poder eliminarse, ya sea porque ya no le es de utilidad al usuario (como por ejemplo que ya no use ese tipo de autobús) o porque se ha equivocado al elegir la red.

Tabla 2: Requisitos funcionales de usuario autenticado

ID	Requisito	Descripción
AU03	Cerrar sesión	Como toda aplicación que dispone de sistemas de autenticación, hay que proveer una manera de poder desconectarse de su cuenta.
AU04	Modificar tiempo de detección de red	Permite al usuario modificar el intervalo de tiempo en el que una red bus es detectada y se inicia el viaje con la misma.
AU05	Modificar intensidad de detecciones de red	Permite al usuario modificar la intensidad que es necesaria para la detección de cuándo el usuario se ha subido al bus (iniciado el viaje) o bajado del mismo (finalizado el viaje).
AU06	Habilitar o deshabilitar el envío automático de datos	Con este requisito, se le da al usuario la opción de no mandar los datos de manera automática, ya que, en ciertas ocasiones, le puede ser imposible o simplemente, por comodidad, prefiere hacerlo él.
AU07	Enviar datos manualmente	Así como se dispone de la opción de deshabilitar el envío automático de datos, debe haber otra que permita al usuario enviar los datos de manera manual.
AU08	Eliminar datos del dispositivo	Para evitar posibles problemas de memoria en el dispositivo, se ha añadido la opción de eliminar los viajes realizados del dispositivo.
AU09	Modificar cuenta	Permite al usuario modificar sus credenciales, como su nombre, correo electrónico o su contraseña.
AU10	Modificar viaje en curso	Aunque el dispositivo móvil intentará en todo momento ser lo más automático posible con la detección de viajes, no es fiable al 100%, por lo tanto, se ha añadido la funcionalidad de que el usuario pueda modificar ciertos parámetros del viaje que está realizando en ese instante.
AU11	Visualizar estadísticas	Permite al usuario visualizar sus estadísticas personales: viajes totales, viajes por día, línea más usada, tiempo medio de viaje, trayecto más largo y contaminación evitada.
AU12	Filtrar estadísticas	Permite al usuario filtrar las estadísticas por fecha.
AU13	Visualizar mapa de rutas recientes	El usuario podrá visualizar en un mapa, sus últimas rutas realizadas.
AU14	Mostrar detalles del viaje	El usuario podrá visualizar los detalles de uno de los viajes que haya realizado recientemente.
AU15	Borrar cuenta	Permite al usuario borrar su cuenta permanentemente del sistema. Sus datos de viaje seguirán permaneciendo intactos en el servidor, pero ya no estarán asociados a él.

Tabla 3: Requisitos funcionales de usuario autenticado (Continuación)

3.2. Requisitos funcionales de la aplicación web

A partir de los requisitos funcionales de la aplicación Android, se han obtenido los de la aplicación web (ver Tabla 4). Estos, al contrario que con la aplicación Android, no van asociados a ninguna categoría en especial.

ID	Requisito	Descripción
WA01	Mostrar estadísticas generales	La aplicación web mostrará las estadísticas generales del sistema: número de viajes totales, media de todos los viajes por día, línea más usada por los usuarios, viaje más largo, duración media de viaje, contaminación evitada, actualización media de línea en los autobuses, bus más antiguo sin actualizarse, trayecto de viaje más largo, trayecto de viaje medio y día más popular de la semana.
WA02	Filtrar estadísticas generales	La aplicación web permitirá filtrar las estadísticas en distintos intervalos de tiempo ya establecidos, así como personalizados por el usuario.
WA03	Enviar correos	La aplicación web será capaz de enviar correos por ciertos motivos, como por ejemplo para confirmar la cuenta de un usuario o para reiniciar su contraseña.

Tabla 4: Requisitos funcionales de la aplicación web

3.3. Requisitos no funcionales

Al contrario que con los requisitos funcionales, estos determinan restricciones o condiciones en el sistema que se ejecuta. En la Tabla 5 se puede observar la lista extraída de todas estas restricciones.

ID	Requisito	Descripción
NFR01	Usabilidad	La aplicación debe ser lo más intuitiva y cómoda posible.
NFR02	Mantenimiento	Las partes del código más complejas, deberán estar bien documentadas, además de estar bien estructurado y espaciado para facilitar la lectura del mismo.
NFR03	Documentación	Se incluirá un manual de usuario entendible por todo tipo de personas.
NFR04	Portabilidad	La aplicación, inicialmente, se podrá usar únicamente desde Android.
NFR05	Almacenamiento	Los datos de los usuarios deben ser almacenados en una base de datos.
NFR06	Interfaz	La interfaz debe ser, en lo mayor posible, agradable e intuitiva.
NFR07	Seguridad	La comunicación tiene que ser HTTPS, debido a que se maneja información privada como contraseñas
NFR08	Rendimiento	La interacción entre la aplicación Android con el servidor debe ser, dentro de lo posible, rápida

Tabla 5: Requisitos no funcionales

3.4. Actores

Después del análisis de requisitos, la identificación de los actores es, en parte, trivial, ya que estos actúan como roles en el sistema. En total se han detectado tres actores en todo el sistema:

- **Usuario**: representa a cualquier persona que no esté autenticada en el sistema, con o sin cuenta creada.
- **Usuario autenticado**: representa al usuario que ha introducido sus credenciales en el sistema.
- **Sistema**: representa el servidor centralizado el cual se encarga de recibir los datos de los usuarios.

3.5. Casos de uso de la aplicación

En esta sección se recoge la especificación de los casos de uso y se ilustra el diagrama (ver Figura 1). Debido a su extensión, la especificación no abarcará todos los casos de uso, sino los más notables.

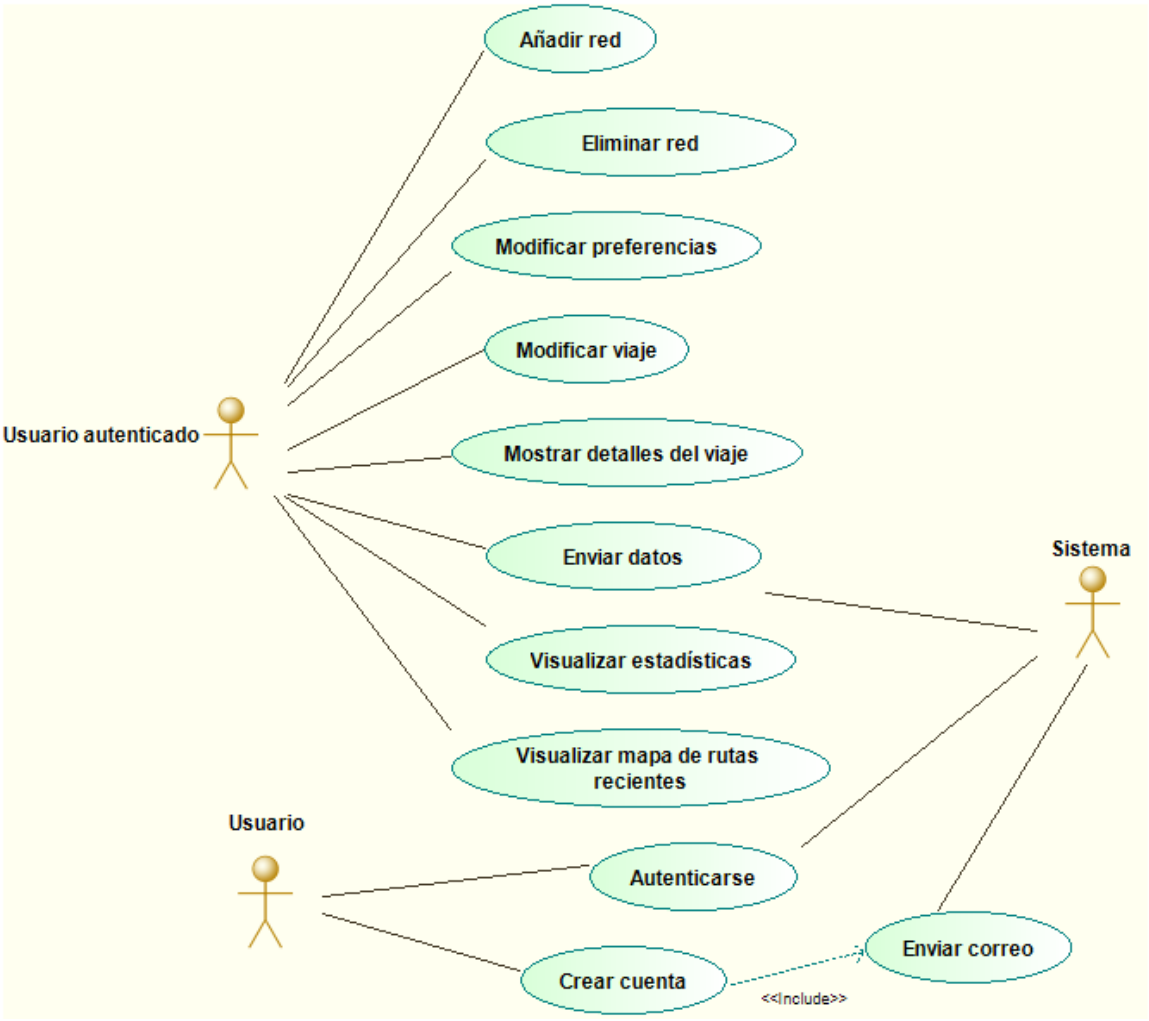

Figura 1: Diagrama de casos de uso

Después de ver el diagrama de casos de uso, se procede a la especificación, detallando en lo posible los escenarios de éxito, así como todas las extensiones de error posibles.

Autenticación en el sistema

Identificador: UC1
Actores: Usuario, sistema
Requisitos asociados: NAU02
Descripción: Autentica un usuario en el sistema.
Precondiciones:

- El usuario debe tener creada una cuenta en el sistema.

Escenario principal de éxito:

1. El usuario abre la aplicación.
2. La aplicación le muestra al usuario la interfaz de autenticación.
3. El usuario introduce sus credenciales en la aplicación.
4. El sistema comprueba que las credenciales existen y son correctas.
5. La aplicación muestra la interfaz principal de inicio y prepara todos los escaneos asociados a la cuenta.

Extensiones:

4.a.1. Las credenciales introducidas por el usuario no existen en el sistema.
4.a.2. La aplicación muestra un mensaje de error para notificar al usuario.
4.b.1. La aplicación no puede conectarse a internet para acceder al servidor.
4.b.2. La aplicación muestra un mensaje de error para notificar al usuario.

Añadir red

Identificador: UC2
Actores: Usuario autenticado
Requisitos asociados: AU01
Descripción: Añade una red para que sea detectada como red de autobús.
Precondiciones:

- El usuario debe estar autenticado en su cuenta.

Escenario principal de éxito:

1. El usuario pulsa en el botón de “Añadir nueva red”.
2. La aplicación muestra una ventana para que el usuario introduzca el nombre de la red.
3. El usuario introduce el nombre de la red deseada.
4. La aplicación guarda el SSID.

Extensiones:

2.a.1. El usuario cancela la operación.
3.a.1. El usuario no introduce el nombre de la red.
3.a.2. El sistema no realiza ninguna operación.

Eliminar red

Identificador: UC3
Actores: Usuario autenticado
Requisitos asociados: AU02
Descripción: Elimina la información de una red de las preferencias para que ya no sea detectada como tal.
Precondiciones:

- El usuario debe estar autenticado en su cuenta.

Escenario principal de éxito:

1. El usuario pulsa en el botón de “Eliminar” de la red en la lista de redes.
2. La aplicación muestra un mensaje de confirmación.
3. El usuario pulsa aceptar.

4. La aplicación elimina el SSID.

Extensiones:

3.a.1. El usuario pulsa cancelar.

Modificar preferencias

Identificador: UC4
Actores: Usuario autenticado
Requisitos asociados: AU04, AU05, AU06, AU08,
Descripción: Modifica las opciones de detección de viaje.
Precondiciones:

- El usuario debe estar autenticado en su cuenta.

Escenario principal de éxito:

1. El usuario abre el menú lateral y pulsa en el botón de “Opciones”.
2. La aplicación muestra la interfaz de opciones.
3. El usuario selecciona la opción que quiere modificar.
4. El sistema muestra una ventana con la opción para modificar.
5. El usuario modifica la opción y pulsa aceptar.
6. El sistema guarda las preferencias.

Extensiones:

3.a.1. El usuario no selecciona ninguna opción.
5.a.1. El usuario pulsa cancelar.

Visualizar estadísticas

Identificador: UC5
Actores: Usuario autenticado
Requisitos asociados: AU11
Descripción: Muestra al usuario las estadísticas de su cuenta.
Precondiciones:

- El usuario debe estar autenticado en su cuenta.

Escenario principal de éxito:

1. El usuario abre el menú lateral y pulsa en el botón de “Ver estadísticas”.
2. La aplicación recoge la información del servidor web.
3. La aplicación muestra las estadísticas al usuario.

Extensiones:

2.a.1. La aplicación falla en recoger la información del servidor web.
2.a.2. La aplicación muestra un mensaje de error de la causa.

Visualizar mapa de rutas recientes

Identificador: UC6
Actores: Usuario autenticado
Requisitos asociados: AU12

Descripción: Muestra los 10 últimos viajes en una ruta sobre un mapa.
Precondiciones:

- El usuario debe estar autenticado en su cuenta.
- El usuario debe habe realizado, al menos, un viaje reciente.

Escenario principal de éxito:

1. El usuario abre el menú lateral y pulsa en el botón de “Ver mapa”.
2. La aplicación recoge toda la información de la base de datos local.
3. La aplicación muestra el mapa vacío con las rutas en una lista.
4. El usuario selecciona una ruta por su fecha.
5. La aplicación pide la ruta al servidor de Google.
6. La aplicación dibuja la ruta en el mapa.

Extensiones:

2.a.1. La aplicación falla en recoger la información de la base de datos local.
2.a.2. La aplicación muestra un mensaje de error de la causa.
4.a.1. El usuario no selecciona ninguna ruta.
5.a.1. La petición al servidor de Google falla.
5.a.2. La aplicación muestra un mensaje de error de la causa.

Envío de datos

Identificador: UC7
Actores: Usuario autenticado, sistema
Requisitos asociados: AU07
Descripción: Envía los viajes realizados por el usuario al servidor.
Precondiciones:

- El usuario debe estar autenticado en su cuenta.
- El envío de datos automático debe estar deshabilitado.

Escenario principal de éxito:

1. El usuario abre el menú lateral y pulsa en el botón de “Opciones”.
2. La aplicación muestra la interfaz de opciones.
3. El usuario selecciona la opción de “Enviar datos”.
4. La aplicación muestra una ventana de progreso mientras envía los datos al servidor.

Extensiones:

3.a.1. El usuario no selecciona ninguna opción.
4.a.1. El usuario cancela la operación.
4.a.2. La aplicación cierra la ventana de progreso y cancela la operación.

Crear cuenta

Identificador: UC8
Actores: Usuario, sistema
Requisitos asociados: NAU01
Descripción: Crea una nueva cuenta en el sistema.
Precondiciones:

- Ninguna.

Escenario principal de éxito:

1. El usuario abre la aplicación.
2. La aplicación muestra la interfaz de autenticación.
3. El usuario pulsa en el botón de “Crear cuenta”.
4. La aplicación muestra la interfaz de crear cuenta.
5. El usuario introduce sus datos y pulsa en aceptar.
6. La aplicación comprueba que los datos son correctos y envía un correo de confirmación.
7. El usuario confirma su cuenta.

Extensiones:

5.a.1. El usuario aborta la acción.

6.a.1. La aplicación detecta que los datos no son correctos.

6.a.2. La aplicación muestra un mensaje de error al usuario.

Modificar viaje

Identificador: UC9

Actores: Usuario autenticado

Requisitos asociados: AU10

Descripción: Modifica los valores de un viaje en curso, como por ejemplo, la línea en la que está transitando el autobús.

Precondiciones:

- El usuario debe tener una cuenta en el sistema.
- El usuario debe estar realizando un viaje.

Escenario principal de éxito:

1. El usuario abre la aplicación mediante la notificación de Android.
2. La aplicación muestra la interfaz con los datos iniciales del viaje.
3. El usuario modifica los datos y pulsa aceptar.
4. La aplicación actualiza el viaje.

Extensiones:

3.a.1. El usuario cierra la aplicación.

Mostrar detalles del viaje

Identificador: UC16

Actores: Usuario autenticado

Requisitos asociados: AU13

Descripción: Muestra los detalles de uno de los últimos viajes del usuario.

Precondiciones:

- El usuario debe tener una cuenta en el sistema.
- El usuario debe haber realizado, al menos, un viaje recientemente.

Escenario principal de éxito:

1. El usuario abre el menú lateral y pulsa en el botón de “Ver mapa”.

2. La aplicación recoge toda la información de la base de datos local.
3. La aplicación muestra el mapa vacío con las rutas en una lista.
4. El usuario selecciona una ruta por su fecha.
5. La aplicación pide la ruta al servidor de Google.
6. La aplicación dibuja la ruta en el mapa.
7. El usuario pulsa en “Mostrar detalles”.
8. La aplicación muestra al usuario los detalles del viaje seleccionado.

Extensiones:

2.a.1. La aplicación falla en recoger la información de la base de datos local.
2.a.2. La aplicación muestra un mensaje de error de la causa.
4.a.1. El usuario no selecciona ninguna ruta y pulsa en “Mostrar detalles”.
4.a.2. La aplicación le pide al usuario que seleccione un viaje.
5.a.1. La petición al servidor de Google falla.
5.a.2. La aplicación muestra un mensaje de error de la causa.

Capítulo 4
Diseño

En este capítulo se presentan los diagramas de distribución y de clases UML. Además, se muestran los modelos relacionales de las bases de datos contenidas en la aplicación Android y en el servidor web.

4.1. Diagrama de distribución

A continuación, se presenta el diagrama de distribución (ver Figura 2) realizado para el proyecto. En él, se puede observar la estructura del despliegue que ha tenido el sistema en general.

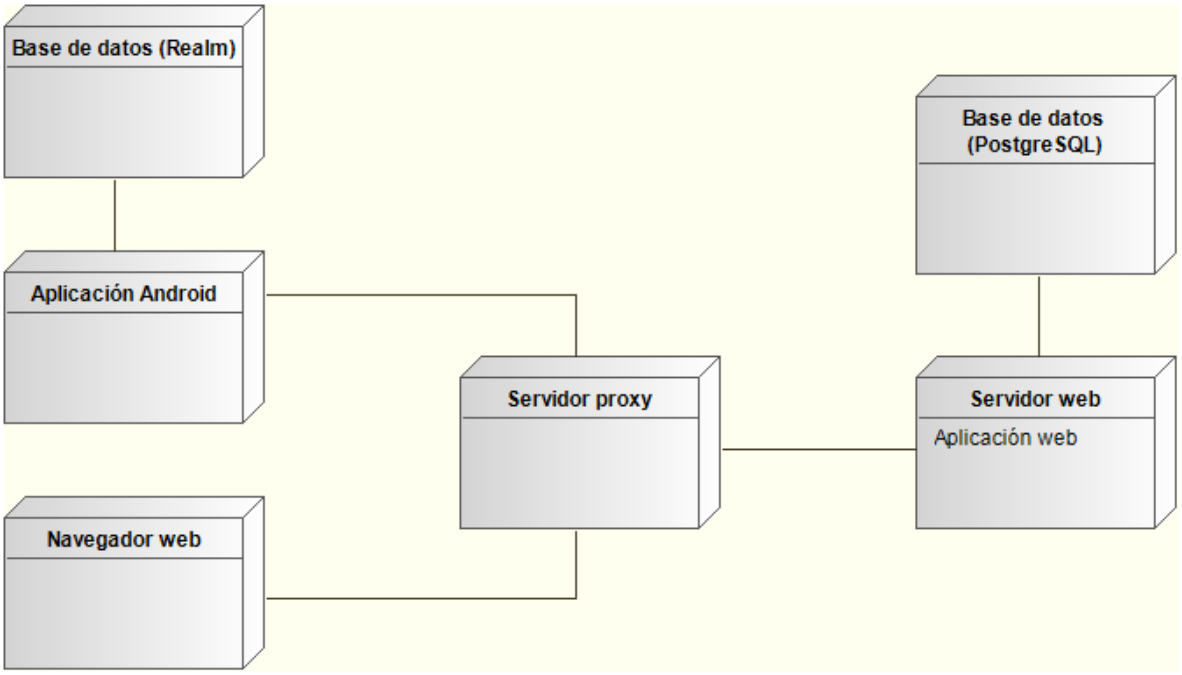

Figura 2: Diagrama de distribución

Las comunicaciones entre la aplicación Android (o el navegador web) y el servidor web, se realizan con peticiones HTTPS en formato JSON, las cuales son redireccionadas por el servidor proxy.

4.2. Modelo relacional de las bases de datos

Para el almacenamiento persistente de los datos, se han tenido que realizar dos modelos de bases de datos distintos, aunque similares: uno para la aplicación Android (ver Figura 3) y otro para el servidor web (ver Figura 4).

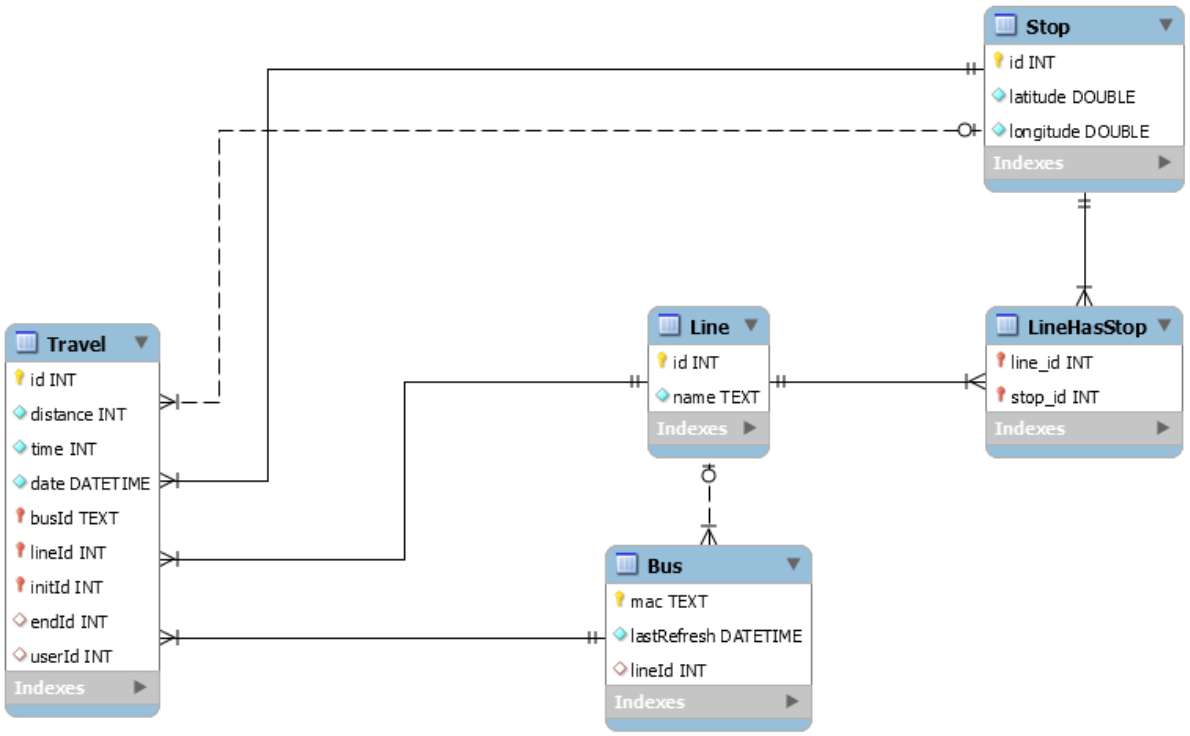

Figura 3: Modelo relacional de la base de datos Android

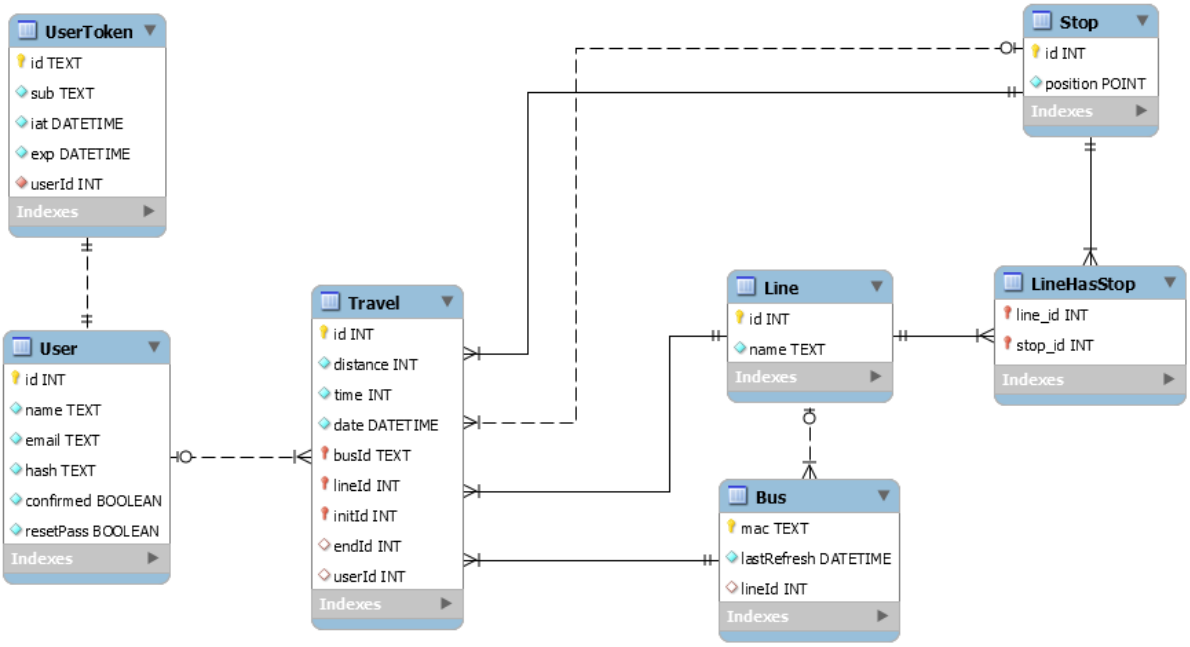

Figura 4: Modelo relacional de la base de datos de la aplicación web

Como se puede observar, se ha estructurado la base de datos de forma en que las aplicaciones puedan acceder o modificar las líneas, las paradas y los autobuses. De esta manera se procede a realizar una breve explicación de las entidades:

- **Travel**: almacena los viajes que realicen los usuarios.
  - **endId**: parada final del viaje. Es una clave foránea a la clave primaria de la tabla Stop.

    - **initId**: parada inicial del viaje. Es una clave foránea a la clave primaria de la tabla Stop.
- **Line**: almacena las líneas de autobús existentes.
- **Bus**: almacena los autobuses detectados por los usuarios.
    - **lastRefresh**: última actualización de la línea del autobús.
- **Stop**: almacena las paradas de autobús existentes.
- **LineHasStop**: entidad de muchos a muchos que relaciona las entidades Line y Stop.
- **User**: almacena las cuentas de los usuarios.
    - **hash**: contraseña del usuario proyectada por un algoritmo *hash* con *salt*.
    - **resetPass**: indica si el usuario ha pedido un reinicio de su contraseña.
- **UserToken**: almacena los *tokens* de refresco que serán usados en el sistema OAuth [12].
    - **sub**: nombre al que va destinado el *token*.
    - **iat**: fecha en la que se ha creado el *token*.
    - **exp**: fecha de expiración del *token*.

4.3. Diagrama de clases

Debido a que la aplicación web ha resultado ser bastante simple, en términos de estructura, no se ha creado ningún diagrama de clases sobre esta. Además, como la extensión que representaba el diagrama de clases entero de la aplicación Android, se muestran únicamente el diagrama (ver Figura 5) sin los atributos y métodos. A continuación, se comentan brevemente algunos aspectos:

- La clase *ScannerService* abstrae la funcionalidad de la detección de viajes.
- La clase *BootLoader* es utilizada para iniciar el servicio de escaneo de la aplicación cuando el sistema Android se encienda.
- La clase *MenuInitializer* es utilizada para inicializar el menú lateral.
- La clase *OAuthUtils* se encarga de realizar todas las funcionalidades que requieran el uso del sistema OAuth.
- La clase *RestUtils* se utiliza para la comunicación con la API REST del servidor, además de tener otros métodos útiles para sincronizar datos.
- Las clases que se encargan de la interfaz están en el espacio de nombres raíz.
- Las clases se han dividido en espacios de nombres acorde a sus usos.

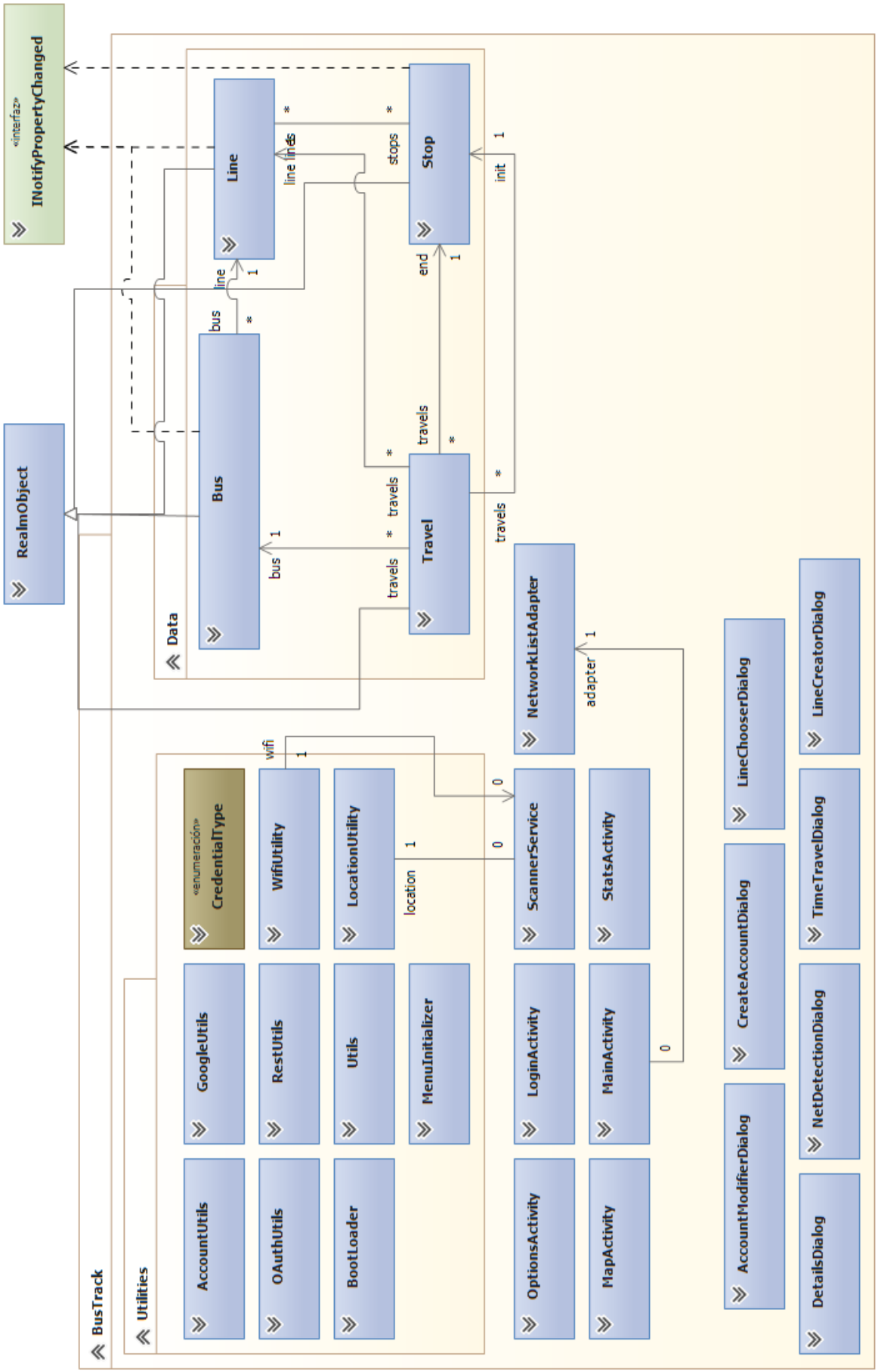

Figura 5: Diagrama de clases

Capítulo 5
Implementación y pruebas

En este capítulo se expone el desarrollo de las partes más importantes del código con las bases de los anteriores capítulos, así como los problemas surgidos y sus soluciones. Por último, se habla de las pruebas realizadas y la estructura de los proyectos de ambas aplicaciones.

5.1. Estructura del proyecto de la aplicación Android

En este apartado se muestra la estructura del proyecto de la aplicación Android. Al estar hecho en Xamarin se ha realizado con Visual Studio (ver Figura 6) es un poco distinto de los típicos proyectos hechos con el Android Studio.

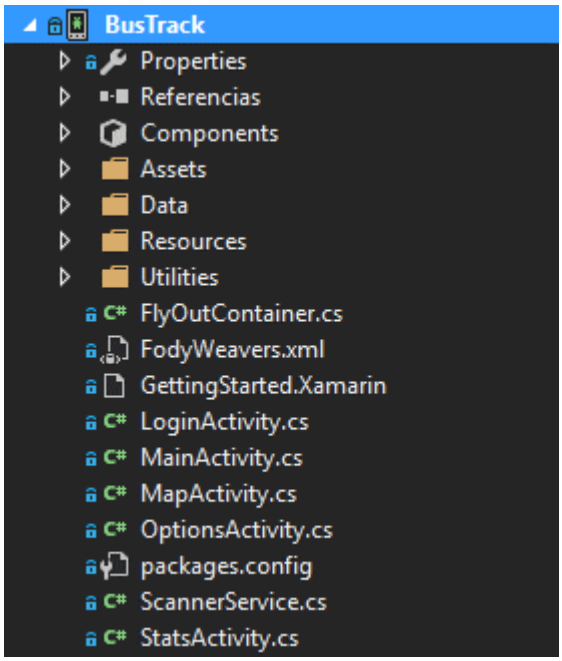

Figura 6: Estructura del proyecto Android

Como se puede observar en la figura, existen varios aspectos que cabe remarcar, como por ejemplo que el Android Manifest se accede a través del apartado Properties. Además, el apartado Components constituye algunas bibliotecas que son de especial importancia debido a que tienen una instalación ligeramente distinta. Por último, en las carpetas Data y Utilities se almacenan los archivos de clase con los espacios de nombres ya nombrados.

5.2. Estructura del proyecto de la aplicación web

A continuación, se describe la estructura del proyecto de la aplicación web (ver Figura 7) en el entorno de desarrollo de Visual Studio:

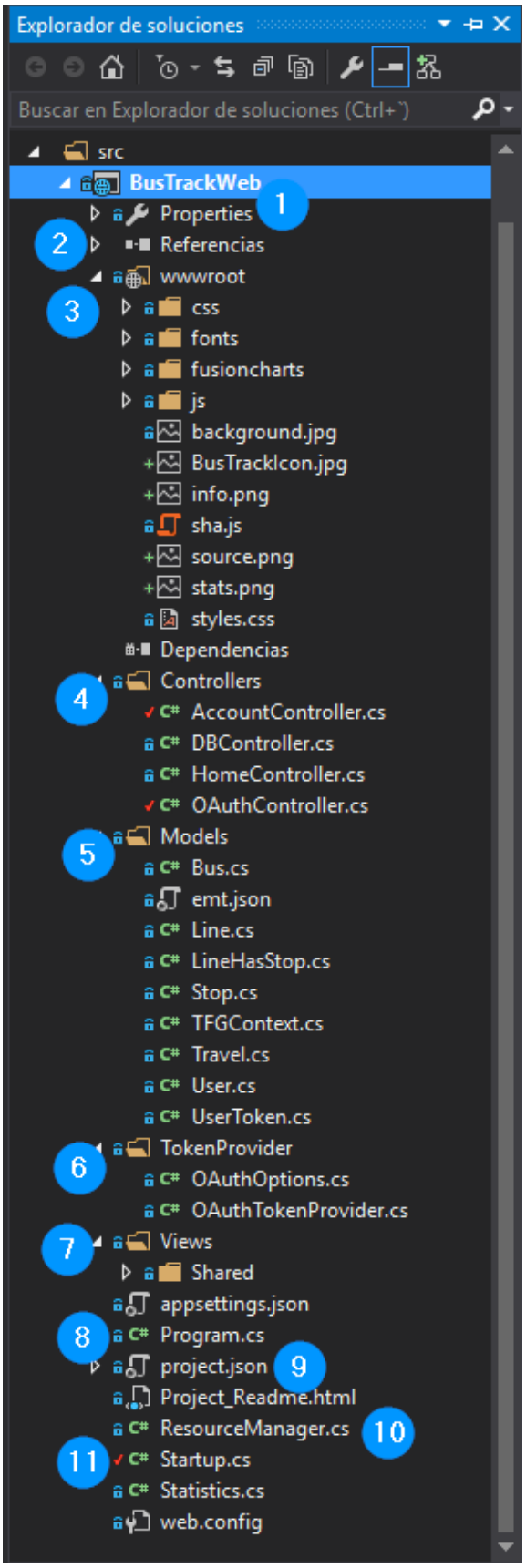

Figura 7: Estructura del proyecto web

1. Propiedades del proyecto. Aquí se pueden modificar las opciones de compilación, así como la configuración del perfil de ejecución.

2. Referencias a los ensamblados de las bibliotecas externas.

3. Carpeta raíz de la web. Aquí se incluyen todos los contenidos estáticos tales como imágenes, archivos CSS, archivos Javascript, etc.

4. Carpeta donde se ubican los controladores de la aplicación web.

5. Carpeta donde están las clases del modelo de datos.

6. Carpeta en donde se encuentran las clases encargadas del sistema OAuth.

7. Carpeta donde se guardan los archivos de vista CSHTML. Actualmente están dentro de la carpeta Shared para que sean accesibles desde cualquier punto de la aplicación web.

8. Program.cs: archivo que contiene el main del proyecto. Aquí se realizan algunas configuraciones cuando se construye lo que actuará como servidor web.

9. project.json: archivo de configuración del proyecto. Guarda todas las referencias, configuraciones, herramientas, etc.

10. ResourceManager.cs: archivo de clases el cual se encarga de que todos los recursos, tales como el archivo emt.json, sean desplegados cuando la aplicación web se ejecute.

11. Startup.cs: archivo de clases que configura la aplicación web. Se eligen los *middleware*, así como las configuraciones de las bibliotecas antes de que la aplicación sea completamente ejecutada.

5.3. Desarrollo del proyecto

Debido a la gran cantidad de tareas que se han implementado a lo largo del desarrollo del proyecto, solamente se hablan de las más importantes o complejas. Además, se comentan los problemas que han surgido a la hora de implementarlas, así como la solución propuesta.

5.3.1. Obtención de líneas y paradas de la EMT

Uno de los principales problemas que se ha encontrado, ha sido la mejor manera de realizar la detección de viajes. Después de mucha investigación, se llegó a la conclusión de que la información debía ser sacada de la propia página de la EMT, ya que, a través de los enlaces, se puede acceder fácilmente a los datos de las líneas y las paradas.

Para esto, se ha hecho uso del *framework* Qt, el cual brinda una gran cantidad de facilidades al desarrollador para todo tipo de situaciones. En este caso se han usado las herramientas tanto de navegación como de expresiones regulares tal y como se explica a continuación:

1. Se navega a la página móvil de la EMT y se extraen los enlaces de las líneas con la siguiente expresión regular: **codLinea=\d+**
2. Se navega por cada página de línea y se busca su nombre con esta expresión: **<h3>[\s\S]+</h3>**
3. Por cada página de línea, se obtienen los enlaces de las paradas con la siguiente expresión regular: **codParada=\d+**
4. Por último, se navega por cada página de parada y se busca su localización con la expresión regular: **lat=-?\d+.\d+&lon=-?\d+.\d+**

Una vez se tienen todas las líneas y paradas, las cuales son 42 y 1043 respectivamente, se crea un archivo JSON (ver Código 1) con los datos guardados para su posterior inclusión en la base de datos del servidor.

```
{
  "lines": [
    {
      "id": Id de la línea,
      "name": "Nombre de la línea",
      "stops": [
        Ids de las paradas
      ]
    }
  ],
  "linesSize": Tamaño de la lista de líneas,
  "stops": [
    {
      "id": Id de la parada,
      "lines": [
        Ids de las líneas
      ],
      "position": "latitud&longitud"
    }],
  "stopsSize": Tamaño de la lista de líneas
}
```

Código 1: Formato JSON

5.3.2. Escaneo de redes

Para el escaneo de redes wifi, se ha hecho uso de lo que se conoce en Android como *BroadcastReceiver*, el cual actúa como un receptor para distintos tipos de acciones. En este caso se ha configurado para que escuche del receptor wifi.

La clase encargada para recibir los datos es *WifiUtility*. En su constructor se obtiene el manejador del wifi del sistema y se registra la clase en el mismo usando un *Intent* para que acepte los resultados del escaneo. Con esto hecho, se puede empezar un escaneo de manera asíncrona muy fácilmente con el método **StartScan**.

Dado que el escaneo de las redes wifi es realizado de manera asíncrona por Android, la lista de resultados debe ser modificada con el uso de monitores para mantener la integridad de la misma. Una vez se realiza la modificación, se notifica al servicio de la aplicación y se utiliza el patrón Observador para enviar la lista de resultados a la UI principal (ver Figura 8).

Figura 8: Interfaz principal

5.3.3. Servicio de escaneo

Por cómo está hecho el sistema operativo Android, una vez que una aplicación es cerrada por el usuario, cualquier operación que se esté realizando es terminada de forma inmediata. Para ello, Android dispone de un tipo de procesos llamados servicios, los cuales están destinados a que se ejecuten en segundo plano incluso si la propia aplicación es cerrada por el usuario. Esto, a su vez, ha sido un problema debido a que en las ocasiones que Android necesite más recursos, este mata a los servicios que más tiempo lleven ejecutándose sin la interacción del usuario. Por ello, ha sido necesario diseñar el servicio de la aplicación para que sea lo más robusto posible a los reinicios forzados que impone Android.

Otro problema que tienen los servicios de Android es que, por defecto, se ejecutan en el hilo de la interfaz, por lo que, si un servicio tarda mucho tiempo en este hilo, la interfaz de Android deja de responder y se preguntará al usuario si desea matar este servicio. Para evitar esto, es necesario iniciar un hilo secundario (ver Código 2) y además es necesario controlar cuando el servicio va a ser detenido por la aplicación, motivo por el cual hay que indicarle al mismo que termine. La aproximación que se ha seguido en este caso ha sido la de enviar una señal de interrupción para que salga de su bucle de ejecución.

```
[Service]
internal class ScannerService : Service
{
	public override void OnCreate()
	{
		base.OnCreate();
		scanner = new Thread(() => ...);
		scanner.IsBackground = false;
		scanner.Name =
"BusTrackScanner";
		scanner.Start();
	}

	public override void OnDestroy()
	{
		base.OnDestroy();
		scanner.Interrupt();
		scanner.Join(100);
	}
}
```

Código 2: Esqueleto del servicio de escaneo

En cuanto al funcionamiento, el servicio usa varios tipos de mecanismos para realizar la comprobación de que el usuario se haya subido o bajado del autobús y haya iniciado o finalizado un viaje:

1. Se inicia un escaneo de redes wifi y se esperan los resultados.
2. Si no se ha iniciado un viaje:
    a. Si alguna de las redes guardadas en las preferencias está en la lista de resultados a una intensidad mayor de la especificada, se guarda la posición y se inicia un temporizador. En el caso de que ya exista la posición en la estructura de datos, no se hace nada.
    b. Si el temporizador ha superado el tiempo límite que se especifica en las preferencias para iniciar el viaje:
        i. Se obtiene la parada más cercana.
        ii. Se busca un autobús con la MAC de la red. En caso de que no exista, se crea un objeto Bus nuevo.
        iii. Se crea el objeto *Travel* en la base de datos con la fecha de inicio, el autobús, la parada y la ID de usuario.
        iv. Si la parada tiene más de una línea o ninguna, mostrar una notificación para que el usuario intervenga.
        v. Si la parada tiene una sola línea, se actualiza el objeto *Travel* con la línea detectada y se muestra una notificación por si no es la correcta.
        vi. Se almacena el estado del servicio de escaneo.
3. Si se ha iniciado un viaje:
    a. Si la red identificada anteriormente no está en los resultados o su intensidad es menor a la especificada en las preferencias, se guarda la posición y se inicia un temporizador.
    b. Si el temporizador ha superado el tiempo límite que se indica en las preferencias para finalizar el viaje. En caso afirmativo, se devuelve la localización guardada.
4. Cuando se ha finalizado un viaje:
    a. Se elimina la notificación creada anteriormente.
    b. Se obtiene la parada más cercana.
    c. Se calcula la distancia entre las dos localizaciones (inicio y fin) con una llamada a la API de Google.
    d. Se actualiza la línea del autobús. Si esta ha cambiado, se intenta enviar la modificación al servidor web.
    e. Se modifica el objeto *Travel* de la base de datos con el tiempo, la distancia y la parada de finalización.
    f. Se envía el viaje al servidor si la preferencia está activa y existe conexión.
    g. Se elimina el último estado del servicio de escaneo.
    h. Se sincronizan las líneas, paradas y autobuses con el servidor si existe conexión.

5.3.4. El BootLoader del servicio

Como se ha indicado en el diagrama de clases, en la aplicación Android existe una clase (ver Código 3) que se encarga de que el servicio de escaneo se ejecute al inicio del sistema operativo solamente si un usuario ha sido autenticado con anterioridad. Esta clase es un *BroadcastReceiver* con la simple funcionalidad de que tiene asociado un *Intent* que configura esta clase para que reciba la notificación de que el sistema Android se ha iniciado.

```
[BroadcastReceiver]
[IntentFilter(new[] { Intent.ActionBootCompleted }, Categories = new[] {
Intent.CategoryDefault })]
internal class BootLoader : BroadcastReceiver
{
      public override void OnReceive(Context context, Intent intent)
      {
            if (intent.Action != null && intent.Action ==
Intent.ActionBootCompleted && OAuthUtils.UserLogged(context))
            {
                  Intent service = new Intent(context,
typeof(ScannerService));
                  service.AddFlags(ActivityFlags.NewTask);
                  service.AddFlags(ActivityFlags.FromBackground);
                  context.ApplicationContext.StartService(service);
            }
      }
}
```

Código 3: Ejecución del servicio al inicio del sistema Android

5.3.5. Almacenamiento de datos

Para la base de datos de la aplicación Android, se ha hecho uso de Realm. Este es un sistema de gestión de bases de datos relacionales (RDBMS) bastante nuevo en el mercado y que difiere un poco sobre los demás de su tipo, ya que no se necesita de ningún script SQL para la creación del esquema de base de datos. Esto facilita la tarea al desarrollador, ya que no hay que estar realizando una trazabilidad entre las consultas SQL preestablecidas y el modelo de clases. Con simplemente hacer que la clase herede de *RealmObject*, ella y todos sus atributos serán almacenados en la base de datos.

Además, en el modelo de clases de la aplicación Android se utiliza el patrón Observador para marcar cuando el objeto ha sido modificado. Con cualquier cambio realizado a esa clase y que no tenga la anotación *Ignore*, se verá reflejado en el esquema de la base de datos.

Para la base de datos a la aplicación web, se ha hecho uso de PostgreSQL. Al igual que Realm, es un RDBMS, aunque este que sí necesita de tener algún script que genere el esquema de la base de datos. Además, para evitar el uso de conexiones y consultas a bajo nivel se ha utilizado la biblioteca de EntityFrameworkCore con la extensión para la base de datos de PostgreSQL: NpgsqlCore.

EntityFrameworkCore dispone de una herramienta para crear el modelo de datos con las tablas de la base de datos ya operativa, pero por ciertos problemas y errores que hubo durante la operación, se hizo de manera manual.

Al contrario de lo que ocurre en Realm, con Npgsql y EntityFrameworkCore, es necesario crear un contexto que se encargue de configurar el modelo de datos cuando se crea, así como indicar la cadena de conexión a la biblioteca y disponer de todas las tablas para un fácil acceso a la base de datos.

En este caso, cuando se configura el modelo de datos, se añaden de manera manual las configuraciones para las claves compuestas e indicar la tabla de muchos a muchos que hay en la base de datos.

Al contrario que sucedía con Realm, las clases del modelo de datos en la aplicación web deben ser objetos sin más, es decir, que su clase base sea *Object* como la de todos los demás.

5.3.6. Uso de un certificado auto-firmado

Uno de los requisitos no funcionales del sistema era el uso de HTTPS. Para esto, es necesario de disponer de un certificado en el servidor. Hoy en día se pueden conseguir certificados de manera gratuita gracias a la fundación Linux, pero debido a que para esto es necesario tener la aplicación web en un servidor externo, ha sido imposible. Por lo tanto, se ha hecho uso de un certificado auto-firmado.

La generación del certificado se ha realizado utilizando la biblioteca más extendida actualmente para SSL: OpenSSL. Para ello, se ha hecho uso de la línea de comandos en Linux para generar un certificado con formato *X.509* de tipo *RSA* de 4096 bits y que sea válido durante 365 días.

Por motivos de seguridad, Android no acepta este tipo de certificados como válidos por defecto, ya que no se puede comprobar su procedencia. Por ello, es necesario realizar modificaciones a la hora de realizar las peticiones web para que esto se vuelva posible (ver Código 4).

```
//Since we are using a self-signed certificate, we have to tell android
that web server certificate is valid
Certificate cert;
CertificateFactory cf = CertificateFactory.GetInstance("X.509"); // Web
server certificate uses X.509 format
using (var stream = context.Assets.Open("nginx.cert")) // Read
certificate from assets
{
     cert = cf.GenerateCertificate(stream);
}

// Add the certificate to the TrustedCerts list
var handler = new AndroidClientHandler();
List<Certificate> certs = new List<Certificate>();
certs.Add(cert);
handler.TrustedCerts = certs;

using (var client = new HttpClient(handler))
{
     ...
}
```

Código 4: Validación del certificado en Android

Para esto, es necesario llevar una copia del certificado del servidor y así indicarle al manejador de Android que es válido añadiéndolo a la lista de certificados de confianza.

5.3.7. Cliente REST

La arquitectura usada a la hora de enviar las peticiones web ha sido REST debido a que está bastante extendida y tiene mucho soporte para implementarla tanto en el cliente como en el servidor.

Uno de los motivos por el cual se ha usado C# ha sido porque, al contrario que en Java, el propio lenguaje ya trae de por sí mecanismos más avanzados, como los métodos asíncronos, y las llamadas a los métodos no son a tan bajo nivel. A continuación, se muestra un ejemplo de código (ver Código 5).

```
public static async Task<Bus> CreateBus(Context context, Bus bus)
{
     var content = new StringContent(JsonConvert.SerializeObject(bus),
Encoding.UTF8, "application/json");

     HttpResponseMessage response = await CallWebAPI("/backend/buses",
CancellationToken.None, context, content, timeout: 10, bearer: true);
     if (response.IsSuccessStatusCode)
     {
          var tmp = await response.Content.ReadAsStringAsync();
          bus = JsonConvert.DeserializeObject<Bus>(tmp);
          bus.synced = true;
     }
     return bus;
}

public static async Task<bool> UpdateBus(Context context, Bus bus)
{
     var content = new StringContent(JsonConvert.SerializeObject(bus),
Encoding.UTF8, "application/json");

     HttpResponseMessage response = await
CallWebAPI($"/backend/buses/{bus.mac}", CancellationToken.None, context,
content, timeout: 10, update: true, bearer: true);
     return response.IsSuccessStatusCode;
}
```

Código 5: Cliente REST

En el código se pueden apreciar las llamadas a la API REST del servidor a través del cliente REST creado en la aplicación Android. En ambos métodos se serializa un objeto Bus al formato JSON y se realiza la llamada a un método genérico que realiza todas las configuraciones para conectarse con el servidor web.

Otra de las cosas apreciables es el uso de la palabra reservada *await*, la cual únicamente se puede usar en los métodos asíncronos y es muy útil cuando es necesaria la concurrencia. Lo que hace exactamente es abstraer del desarrollador los mecanismos de espera y sincronización para que solamente se tenga que concentrar en el código. Además, esto es muy útil cuando se usa en métodos que provienen del hilo principal de la interfaz ya que este mecanismo hace que nunca se congele. Por ejemplo, en el código es usado tanto para la espera de la llamada a la API de la aplicación web como para leer la respuesta en el método de crear el autobús.

La llamada al método **CallWebAPI** realiza la configuración para las peticiones web. Como este método es generalizado, se han incluido bastantes argumentos, de los cuales gran parte ya tienen un valor por defecto:

- **urlPath:** ruta de la URL, excluyendo la URL base. Ejemplo: /backend/buses
- **token:** el *token* de cancelación usado para cancelar la petición.
- **context:** el contexto de Android, el cual se usa para obtener el certificado y acceder a las preferencias.

- **content:** el contenido de la petición web. En el caso de la API REST, será un contenido de texto plano de tipo JSON mientras que, en el caso de las demás llamadas, se usa un contenido de tipo X-FORM-URL-ENCODED. Por defecto, este argumento es nulo y se usará una petición de tipo GET.
- **checkLogin:** indica a la llamada si, antes de realizarla, hay que comprobar si el *token* OAuth sigue siendo válido o no.
- **timeout:** indica el tiempo de *timeout* para la petición.
- **update:** indica si la petición creará una nueva entidad o no, en cuyo caso se usará una petición POST o PUT respectivamente.
- **bearer:** indica si es necesaria la autenticación.

Una vez se ha añadido el certificado como uno de confianza, se configura el cliente para que tenga los siguientes parámetros: un búfer de respuesta de 256 KB, el tiempo de espera dado en el argumento y, si es necesaria la autenticación, el *token* de acceso del usuario si existe.

Por último, se selecciona el tipo de petición a través del contenido a enviar, si hay, y de la variable booleana. Si ocurre alguna excepción en la parte del cliente, se devuelve una respuesta manual con el mensaje de la excepción ocurrida.

5.3.8. Sistema OAuth

Para el sistema de autenticación de usuarios se ha utilizado el estándar OAuth2 (ver Figura 9) con una implementación propia sin que dependa de terceros como Google, Facebook, etc.

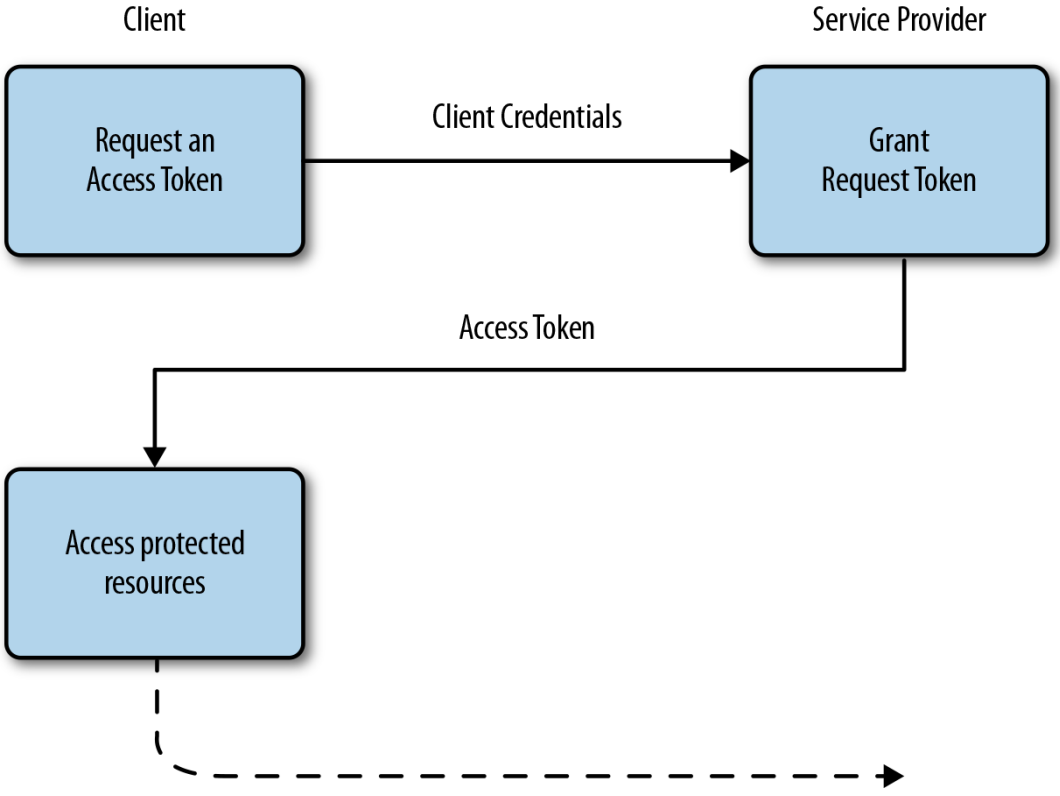

Figura 9: Estándar OAuth2

En todo momento, la comunicación es segura usando el protocolo HTTPS, pero para darle un nivel más de seguridad [13], la contraseña es proyectada por un algoritmo *hash* SHA de 512 bits con una *salt* única (correo electrónico) que, además, también es proyectada por el mismo algoritmo. Todo esto es codificado en formato de base 64.

En la parte del servidor [14], se ha usado el *middleware* llamado JwtBearer que habilita la recepción de *tokens* de autenticación. Para ello, es necesario activar este *middleware* cuando se realiza la configuración de la aplicación web además de añadir el validador de *tokens* con las opciones por defecto de la clase *OAuthOptions* el cual es usado para verificar la firma de estos.

En el sistema OAuth, los *tokens* tienen múltiples atributos que tienen vital importancia [15]. Aunque en este proyecto se han añadido varios adicionales, solamente es necesario remarcar tres:

- **Token de acceso (*access_token*)**: es un *JSON Web Token* (JWT) el cual contiene todos los parámetros de las opciones usadas en el sistema OAuth, además del nombre del usuario como un identificador único. Todo esto es codificado en el formato base 64.
- **Token de refresco (*refresh_token*)**: es un identificador universal único, el cual es la clave primaria de la tabla UserToken. En este proyecto, este *token* expira por defecto en un mes.
- **Vida del token de acceso (*expires_in*)**: es el tiempo de vida restante del *token* de acceso en segundos. En este proyecto, está configurado para que sea de 24 horas.

Además de generar *tokens* de autenticación, también es posible revocar y refrescar los mismos. En el caso del refresco, simplemente se realiza una petición web con el *token* provisto en la generación y, al comprobarse de que sigue siendo válido, vuelve a realizarla con el nuevo JWT. Por otro lado, en el caso de la revocación, se depende del tipo del *token*: si es de acceso, se mete en una lista negra durante 24 horas para que el validador lo rechazo, y si es de refresco, simplemente se elimina de la base de datos.

5.3.9. Realización de estadísticas usando MapReduce

Debido a que algunas estadísticas eran muy complicadas se ha tenido que utilizar una pequeña implementación de un sistema MapReduce. El funcionamiento de este sistema, de manera abstracta, se puede observar en la Figura 10. Este se ha usado expresamente para las siguientes dos estadísticas:

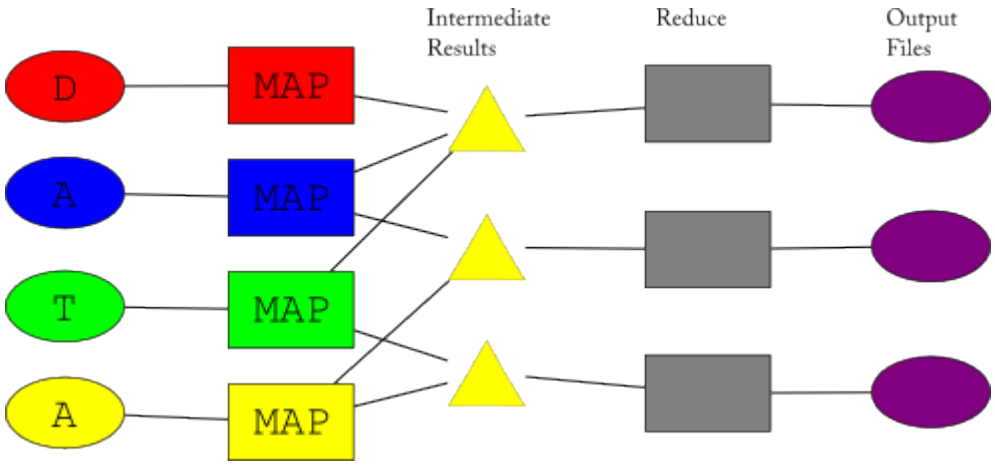

Figura 10: Esquema de MapReduce

- **Viajes por día**. Para mejorar la precisión de esta, se ha tenido que mapear los viajes en fechas y luego reducirlos a números en un diccionario. Por último, se ha hecho la media de todos los valores (ver Código 6).
- **Día más popular de la semana**. En este caso se han mapeado los viajes en su día de la semana para, a continuación, reducirlos a números en un diccionario y, finalmente, coger el valor más alto.

```
private ConcurrentBag<DayOfWeek> travelWBag = null;
private BlockingCollection<DayOfWeek> travelWChunks = null;
private ConcurrentDictionary<DayOfWeek, int> travelWStore = null;

internal int MapReduceTravelsDayWeek()
{
      if (travelWChunks == null || travelWChunks.IsAddingCompleted)
      {
            travelWBag = new ConcurrentBag<DayOfWeek>();
            travelWChunks = new
BlockingCollection<DayOfWeek>(travelWBag);
            travelWStore = new ConcurrentDictionary<DayOfWeek, int>();
      }

      ThreadPool.QueueUserWorkItem((o) =>
      {
            MapWTravels();
      });

      ReduceWTravels();

      return travelWStore.Count > 0 ? (int)travelWStore.Aggregate((a, b)
=> a.Value >= b.Value ? a : b).Key : -1;
}
```

Código 6: Ejemplo de MapReduce

5.3.10. Sistema para enviar correos

Para la confirmación de cuentas y el reinicio de contraseñas, ha sido necesario disponer de un sistema para enviar correos a los usuarios del sistema. Para ello, se ha hecho uso de la biblioteca MailKit. A continuación, se puede observar el funcionamiento de esta para enviar correos con código HTML (ver Código 7). Por motivos evidentes, no se ha incluido la cuenta de correo usada ni su contraseña.

```
var msg = new MimeMessage();
msg.From.Add(new MailboxAddress("BusTrack", "BusTrack@gmail.com"));
msg.To.Add(new MailboxAddress(name, email));
msg.Subject = "Confirmación de cuenta BusTrack";
var html = new BodyBuilder();
html.HtmlBody = $"Por favor, confirma tu cuenta haciendo click en el
enlace:<br/><a href='{url}'>{url}</a>";
msg.Body = html.ToMessageBody();

using (var client = new SmtpClient())
{
      client.Connect("smtp.gmail.com", 587);

      // Disable OAuth2 for this client
      client.AuthenticationMechanisms.Remove("XOAUTH2");
      // Input password or use an alternative mail server
      client.Authenticate("BusTrack", "PASSWORD");
      client.Send(msg);
      client.Disconnect(true);
}
```

Código 7: Envío de correos

5.4. Pruebas

Debido a que en los emuladores Android es imposible simular localizaciones y redes wifi, no se han podido generar pruebas unitarias para comprobar que todo sea correcto. Por esto, se han hecho procesos de *debugging* y pruebas de campo real (ver Figura 12 y Figura 11).

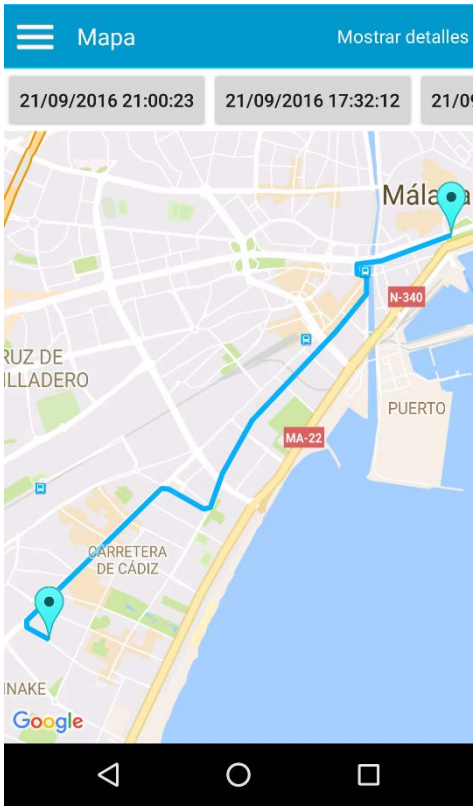

Figura 11: Mapa de rutas

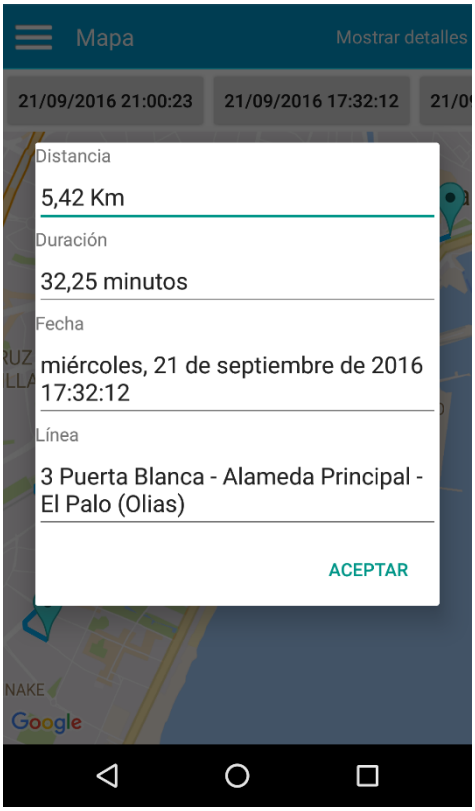

Figura 12: Detalles de la ruta

Para ello se han realizado dos viajes en autobuses que contienen redes wifi. Con esto se aprecia que el sistema funciona de forma casi automática, ya que a la hora de iniciar una ruta en una parada donde llegue más de una sola línea, el usuario tendrá que corregir la línea por la que viaja. Esto se soluciona con más usuarios para que la aplicación tenga los datos sobre los autobuses en tiempo real.

Por otro lado, la aplicación web se ha probado utilizando un cliente REST externo. Para ello se han realizado distintos tipos de llamadas a la API REST con diferentes parámetros en los enlaces o en el contenido. A continuación, se muestran algunas capturas (ver Figura 13, Figura 14 y Figura 15) en donde se puede apreciar el comportamiento de la API REST.

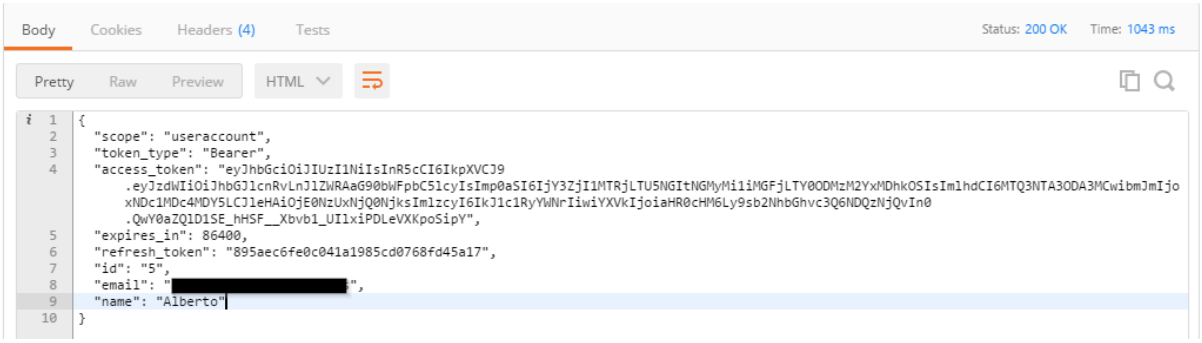

Figura 13: Generación de token mediante usuario y contraseña

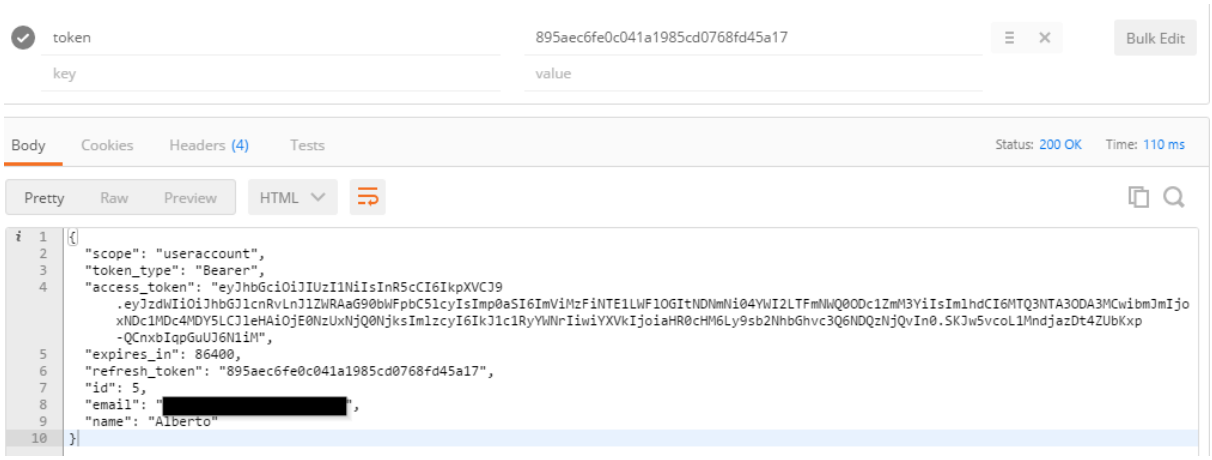

Figura 14: Refresco de token

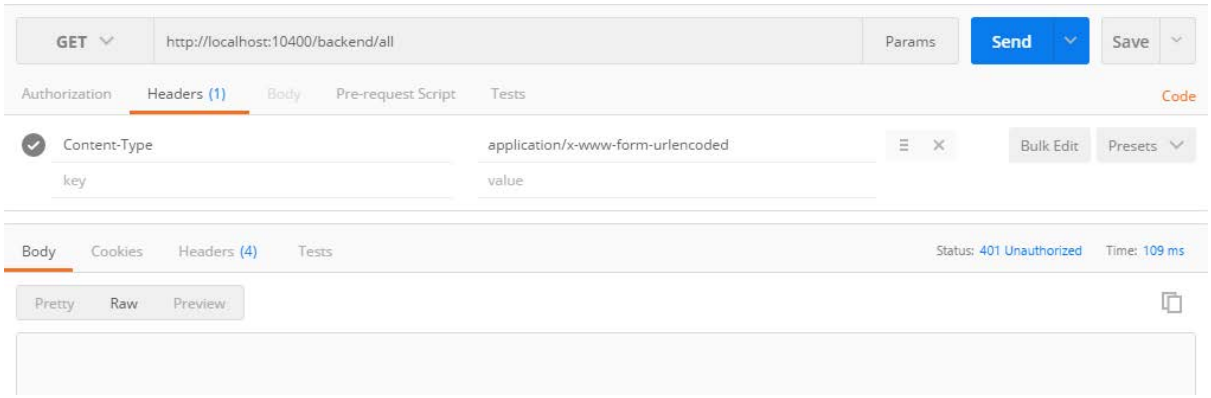

Figura 15: Acceso no autorizado a una API protegida

Capítulo 6
Conclusiones y líneas futuras

6.1. Conclusiones

La movilidad inteligente se ha convertido en uno de los aspectos más importantes en las ciudades inteligentes. El transporte público parece ser una de las mejores soluciones para ofrecer una movilidad más eficiente y sostenible dentro de las ciudades. En este trabajo nos hemos centrado en el transporte mediante autobuses urbanos. Para remarcar los beneficios del transporte público se ha propuesto BusTrack, un sistema que recolecta información sobre los viajes que realizan los usuarios.

Para no forzar al usuario a mantener en todo momento el GPS encendido, se ha decidido identificar las líneas de autobuses y a los propios vehículos empleando las redes wifi que ofrecen. Cabe destacar que se ha utilizado información real de la EMT para mejorar la detección. La información obtenida permite generar estadísticas sobre el uso de este tipo de transporte que es de gran utilidad tanto para los usuarios como para las instituciones.

Para ofrecer esto se ha desarrollado una aplicación móvil y un servidor, los cuales se encargan de recolectar los datos y de generar estadísticas respectivamente. En el desarrollo ha habido una serie de desafíos para llegar a la solución propuesta como la extracción de datos del sistema de líneas y paradas de la EMT o como el sistema de autenticación OAuth2 para los dispositivos móviles. Otra de las cosas que ha supuesto un reto, ha sido las pruebas de la aplicación Android en entornos reales puesto que el número de autobuses disponibles con redes wifi es bastante bajo.

6.2. Líneas futuras

Tras la realización del TFG se han recopilado algunas líneas futuras:

- **Exportar la aplicación a IOS y Windows Mobile.** Aunque Android se lleva la mayor parte del mercado, hay gente que no dispone de uno, por lo tanto, sería atractivo el exportar la aplicación para los demás sistemas.
- **Agregar más estadísticas.** En este proyecto se ha intentado dar todas las estadísticas posibles en lo que ha dado tiempo, así que se puede haber quedado algo sin ponerse o, simplemente, se pueden añadir otras estadísticas que puedan ser interesantes.
- **Adaptar el sistema para otros tipos de transporte público.** Debido a que el transporte público no solo abarca los autobuses, se podría extender la funcionalidad para así mejorar la movilidad por la ciudad.

- **Extender el sistema OAuth.** Aunque bien es cierto que es muy atractivo el hecho de tener un sistema propio, se puede extender para aceptar usuarios de otras plataformas, como por ejemplo Google, y así evitar tener que estar introduciendo contraseñas y mejorar la comodidad del usuario.
- **Mejorar la obtención de líneas y paradas.** De por sí, el sistema es muy exacto, pero tiene el inconveniente de que se ha realizado a parte debido al tiempo que tarda en recoger todos los datos. Una de las cosas que se podría hacer, es incluirlo en la aplicación web con una tarea periódica en segundo plano.
- **Añadido de un histórico.** La generación de estadísticas se ha realizado acorde a lo pedido, pero también cabe la posibilidad de que se requiera una elaboración en un intervalo de tiempo personalizado.
- **Herramientas de *Business Intelligent*.** Con la recolección de estadísticas, se pueden añadir herramientas de *Business Intelligent* para extender el sistema y detectar, por ejemplo, patrones de conducta.

Referencias

[1] Dameri, R. P.-S. (2014). Smart City. Springer.

[2] hibridosyelectricos. (4 de Mayo de 2015). Obtenido de http://www.hibridosyelectricos.com/articulo/sector/indicadores-contaminacion-automovil-2014/20150504180747009369.html

[3] Organization, W. H. (2011). Transporte Urbano y Salud. Dominik.

[4] Documentación Xamarin. (s.f.). Obtenido de https://developer.xamarin.com/recipes/

[5] Documentación C#. (s.f.). Obtenido de https://msdn.microsoft.com/es-es/library/hh846503.aspx

[6] Documentación Bootstrap. (s.f.). Obtenido de http://getbootstrap.com/getting-started/

[7] Documentación FusionCharts. (s.f.). Obtenido de http://www.fusioncharts.com/dev/

[8] Documentación .NET Core. (s.f.). Obtenido de https://docs.asp.net/en/latest/

[9] Documentación Realm. (s.f.). Obtenido de https://realm.io/docs/

[10] Documentación Npgsql. (s.f.). Obtenido de http://www.npgsql.org/doc/index.html

[11] Documentación Qt. (s.f.). Obtenido de http://doc.qt.io/qt-5/

[12] Joudeh, T. (16 de Julio de 2014). bitoftech. Obtenido de http://bitoftech.net/2014/07/16/enable-oauth-refresh-tokens-angularjs-app-using-asp-net-web-api-2-owin/

[13] Hunsaker, C. (24 de Octubre de 2012). stormpath. Obtenido de https://stormpath.com/blog/password-security-right-way

[14] Barbettini, N. (31 de Mayo de 2016). stormpath. Obtenido de https://stormpath.com/blog/token-authentication-asp-net-core

[15] servicenow. (6 de Julio de 2016). Obtenido de http://wiki.servicenow.com/index.php?title=Generating_OAuth_Tokens#gsc.tab=0

Apéndices

A. Manual de usuario de la aplicación Android

El APK de la aplicación es proporcionado en el CD provisto junto a la memoria impresa. Simplemente es necesario copiar el APK en el dispositivo móvil y ejecutar el archivo para proceder con la instalación. Una vez instalada la aplicación Android, nada más ejecutarla se muestra la interfaz de autenticación (ver Figura 16). Para utilizarla, es necesario la creación de una cuenta, por lo que al pulsar sobre “Crear cuenta”, se muestra una ventana (ver Figura 17) donde es necesario introducir las credenciales de la cuenta a crear.

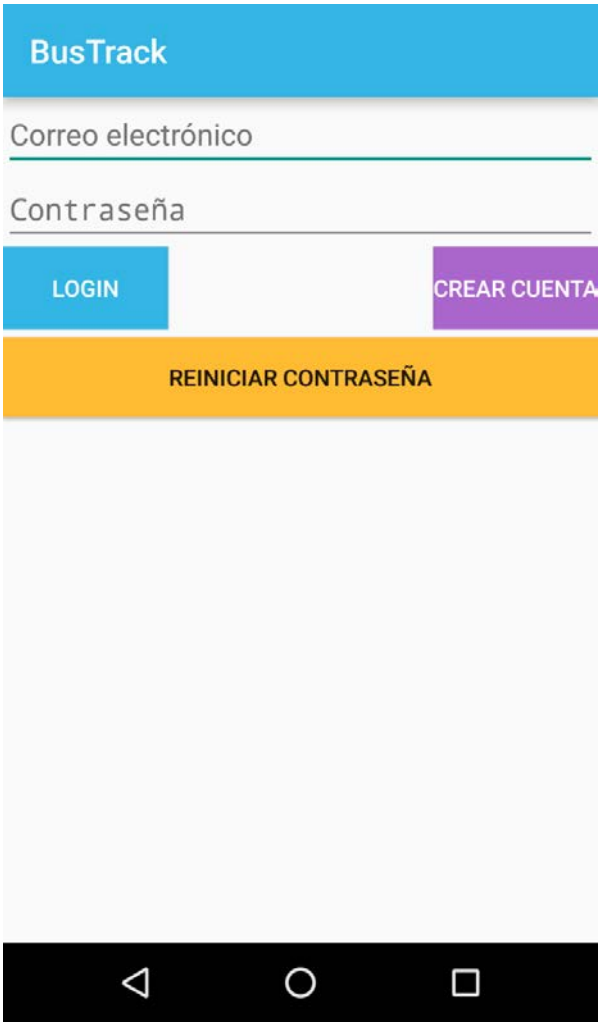

Figura 17: Interfaz de autenticación

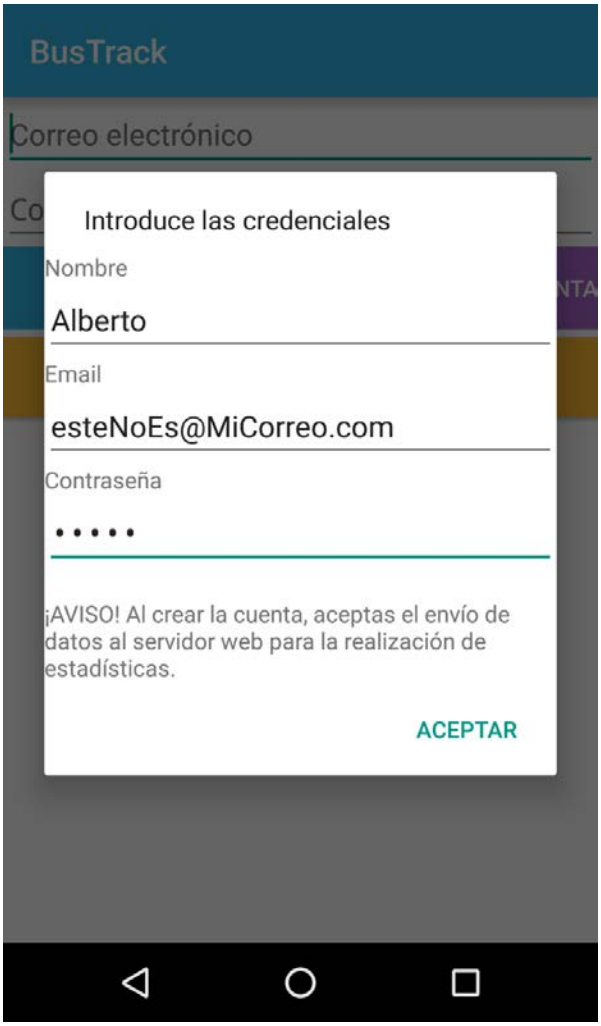

Figura 16: Creación de cuenta

Una vez introducidos los datos, al pulsar en “Aceptar” y el sistema enviará un correo de comprobación como se puede apreciar en la Figura 18. Para confirmar la cuenta, es necesario acceder al enlace facilitado en el correo como se puede apreciar en la Figura 19.

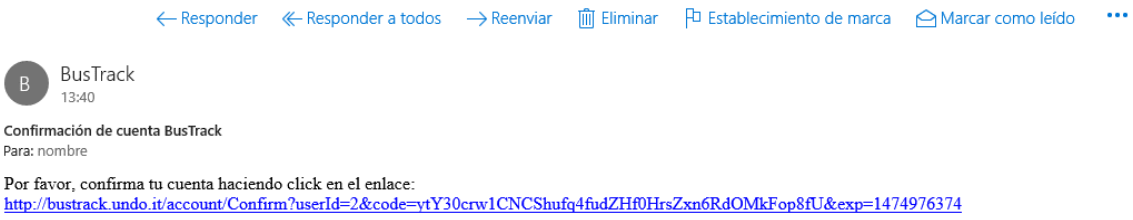

Figura 18: Correo de confirmación

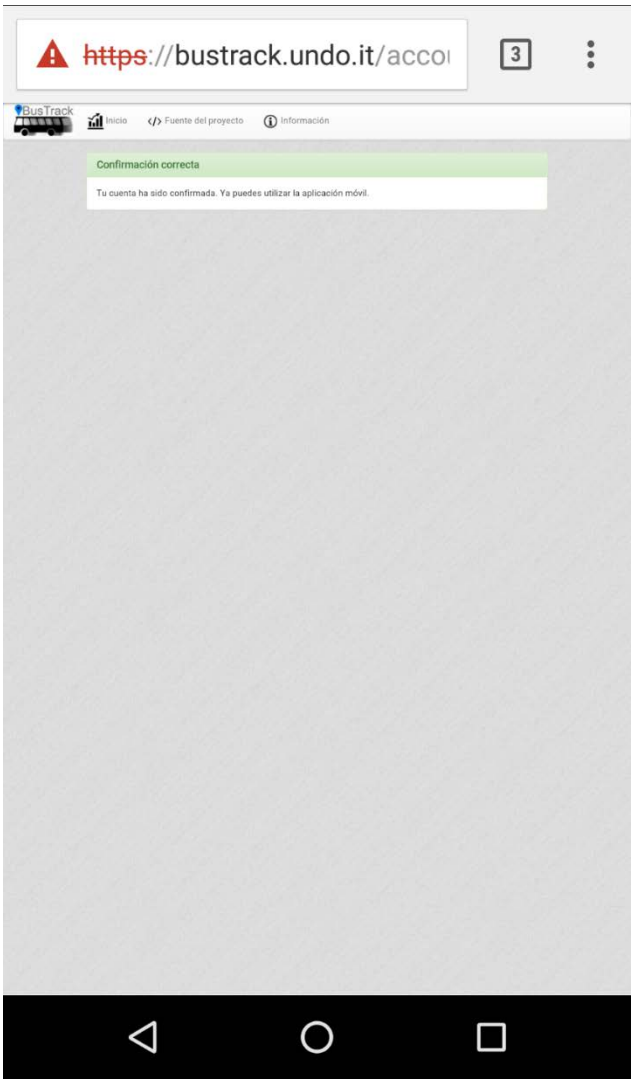

Figura 19: Confirmación correcta

De nuevo en la aplicación y con la cuenta confirmada, es posible autenticarse. Tras esto, se mostrará la interfaz principal de la aplicación como se muestra en la Figura 21.

Figura 21: Interfaz principal

Figura 20: Menú de opciones

En la figura se puede observar que ya hay una red que será detectada por defecto: La red de la EMT. Con esto, los viajes que se realicen en los autobuses de la EMT, serán detectados automáticamente. Si se quiere detectar otro tipo de red, lo mejor será añadirla manualmente cuando se detecte o a través del botón “Añadir nueva red”. Para acceder a las demás opciones, se puede abrir un menú lateral (ver Figura 20).

Dado que en la sección de Implementación y pruebas ya se han facilitado ilustraciones sobre “Ver mapa”, se mostrarán las interfaces de los apartados “Estadísticas recientes” (ver Figura 23) y “Opciones” (ver Figura 24).

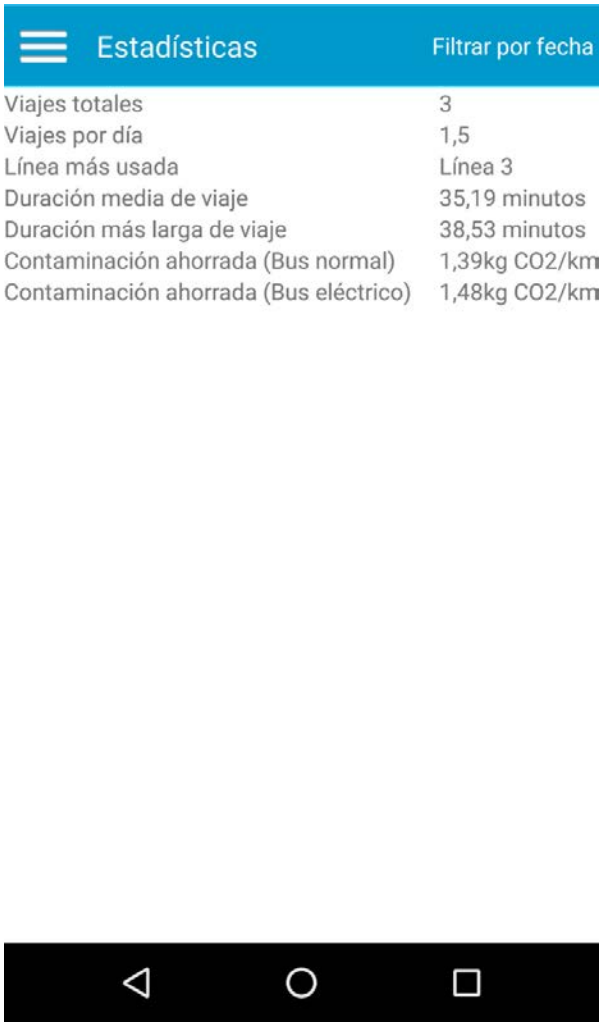

Figura 23: Interfaz de estadísticas personales

Figura 22: Filtrado de estadísticas

En las estadísticas personales se pueden observar diversos datos con la información que se ha extraído de los viajes realizados además de filtrarlos por fechas como se puede apreciar en la Figura 22.

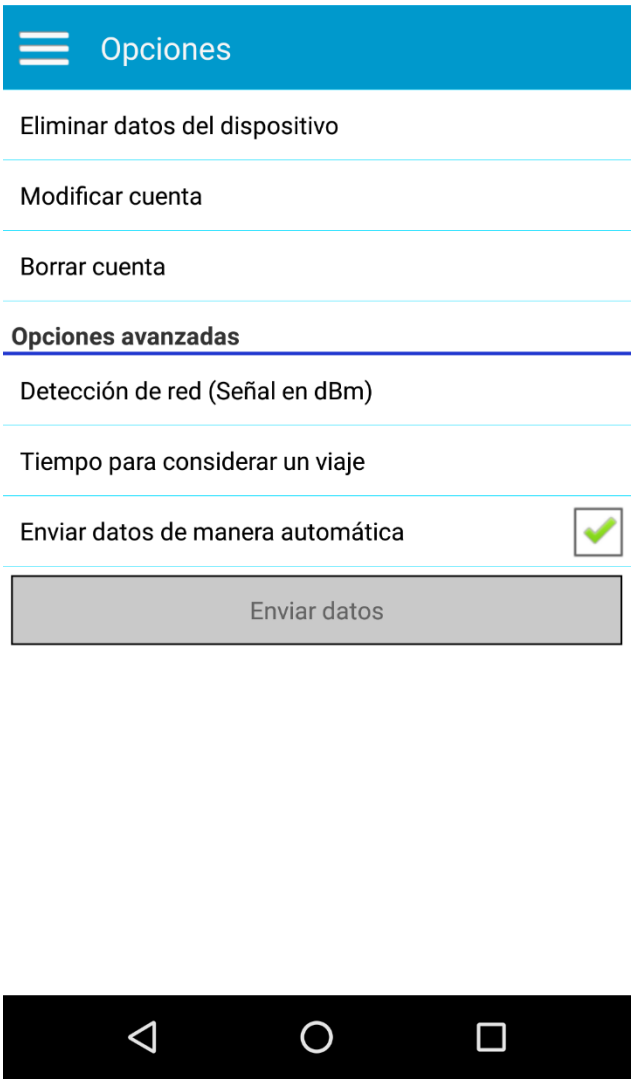

Figura 24: Interfaz de opciones

En la interfaz de opciones se pueden realizar diversas acciones, entre las que se incluyen:

- **Eliminar datos del dispositivo.** Tal y como indica el nombre, esta opción borra los viajes que hay en el dispositivo.
- **Modificar cuenta.** Esta opción permite modificar la cuenta, tal y como se indica en la Figura 25.

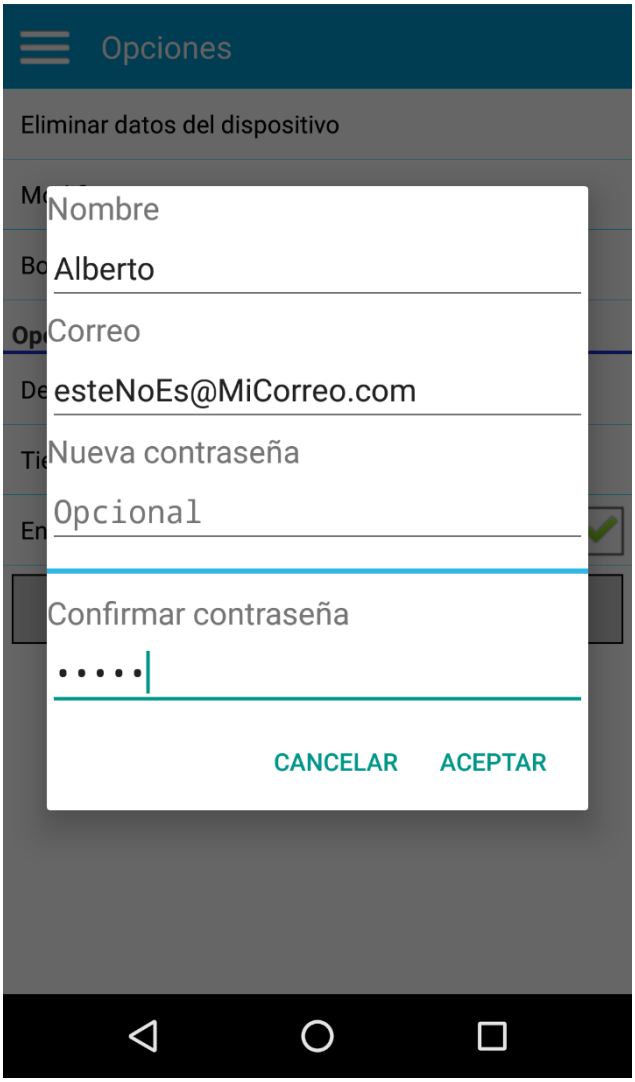

Figura 25: Ventana de modificar cuenta

Para confirmar los cambios, se ha de introducir la contraseña (antigua en el caso de que también se cambie la misma).

- **Borrar cuenta.** Tal y como indica, borra la cuenta del sistema. Para confirmar el borrado, será necesario introducir la contraseña.
- **Detección de red.** Indica la intensidad a la que se tomará en cuenta la subida y la bajada del autobús, por defecto sus valores son 50 y 70 dBm respectivamente (ver Figura 27).

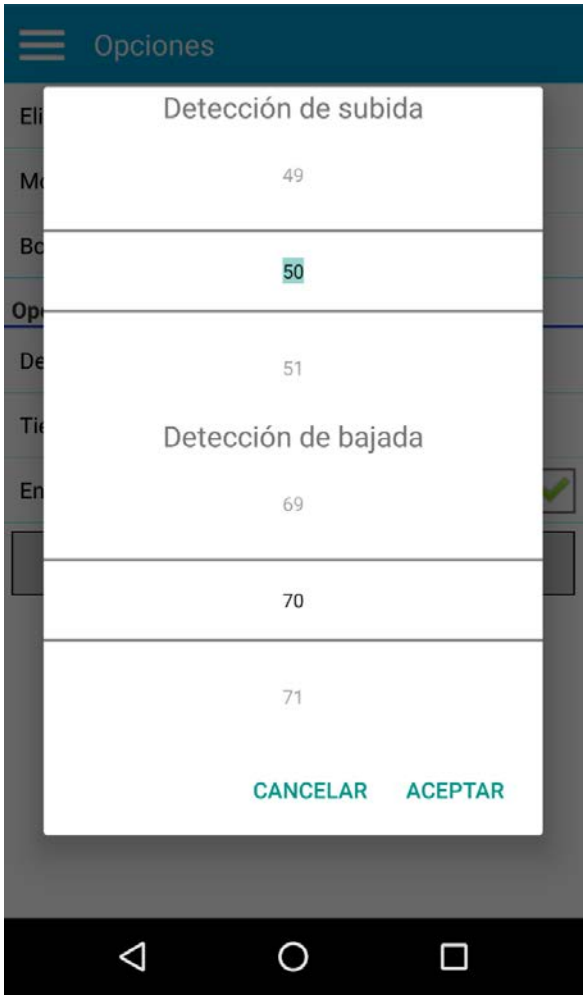

Figura 26: Ventana de detección de red

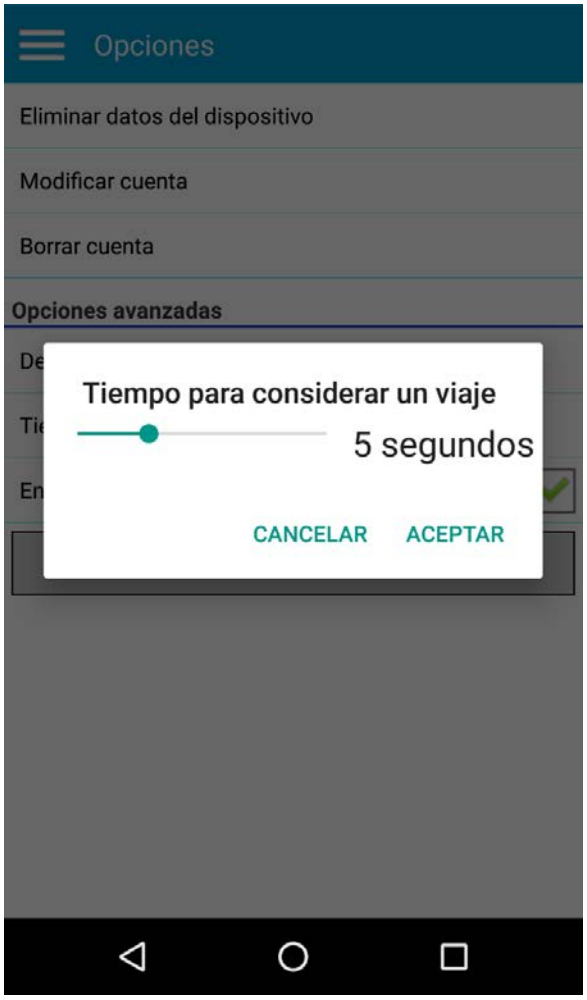

Figura 27: Ventana de tiempo para considerar un viaje

- **Tiempo para considerar un viaje.** Esta opción cambia el tiempo necesario para que se confirme o se finalice el viaje (ver Figura 26).
- **Envío de datos de manera automática.** Indica si la aplicación debe mandar los datos de forma automática o se enviarán cuando el usuario seleccione la opción de “Enviar datos”.
- **Enviar datos.** Envía los datos de forma manual. Esta opción solo está disponible si el envío de datos de manera automática está deshabilitado.

B. Manual de instalación de la aplicación web

A continuación, se realiza el manual de instalación de la aplicación web. La instalación se realizará en una máquina virtual con Linux Mint 18 de 64 bits.

Paso 1 - Instalación de PostgreSQL

Para empezar la instalación, se abre la terminal y se ejecuta el comando de la Figura 28.

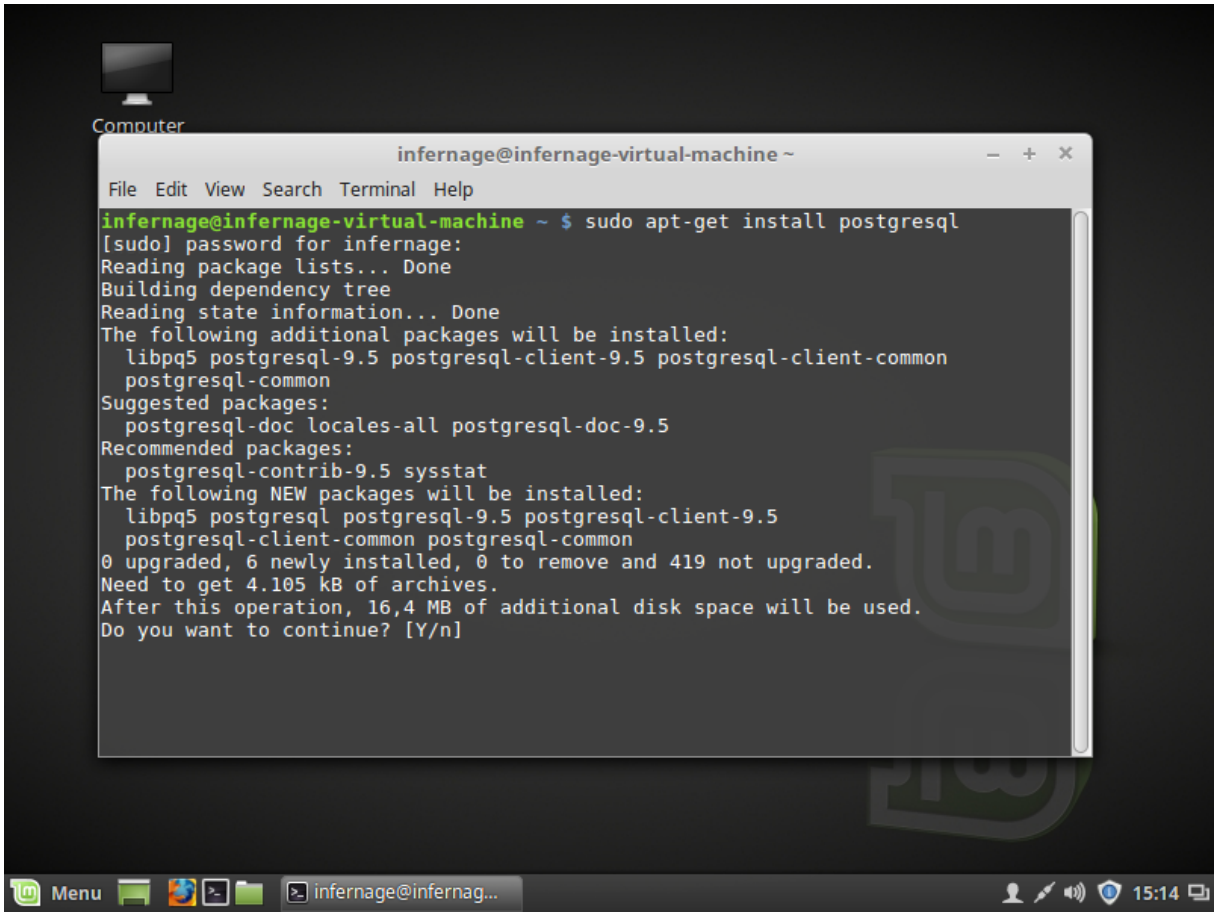

Figura 28: Instalación de PostgreSQL

Tras esto, para tener una interfaz gráfica a la hora de configurar todos los parámetros e instalar el esquema de la base de datos, es necesaria la utilidad pgadmin3. Para ello se instala como procede en la Figura 29.

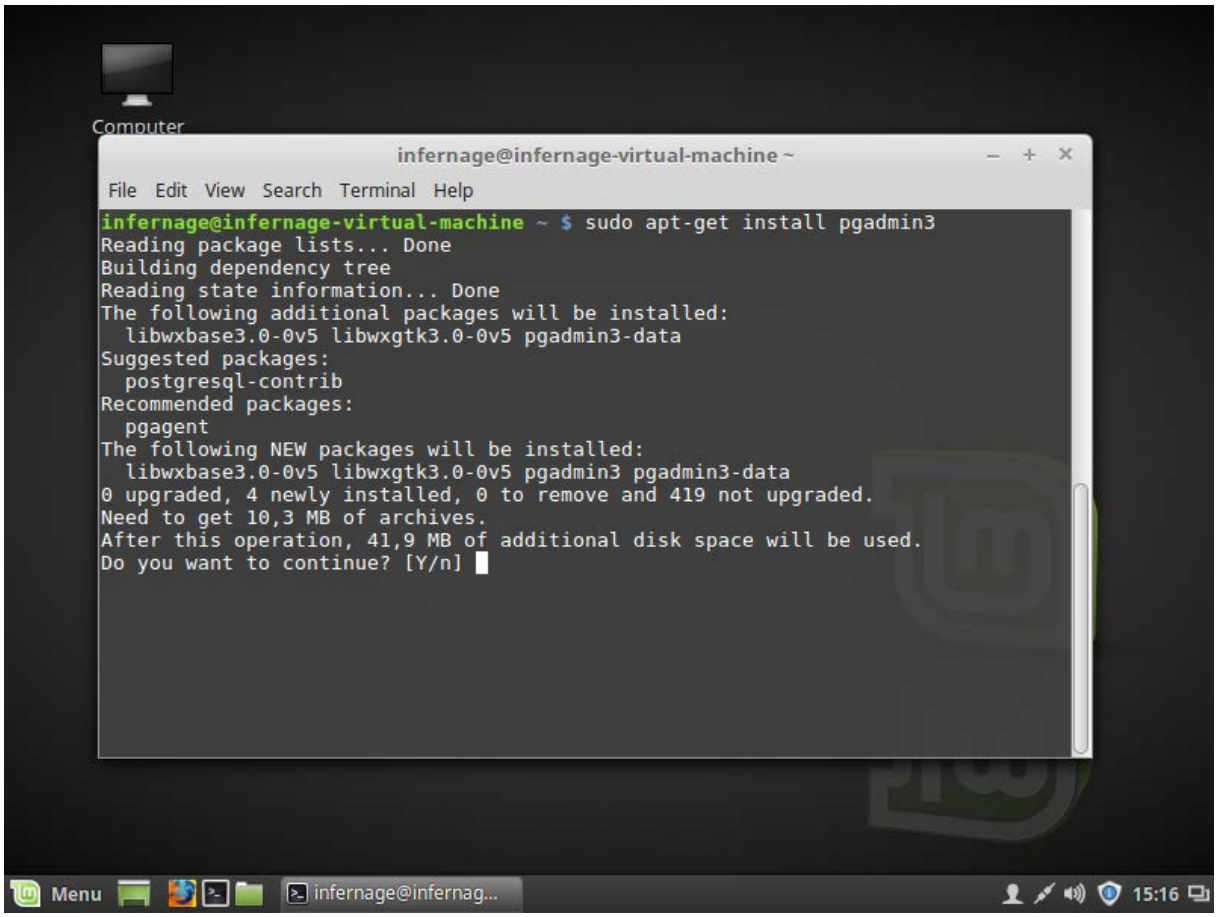

Figura 29: Instalación de pgadmin3

Paso 2 - Instalación de .NET Core

Para instalar .NET Core, es necesario actualizar las fuentes de paquetes, añadiendo la de .NET Core (ver Figura 31). Tras esto, se procede a la instalación del *framework* con el comando que aparece en la Figura 30.

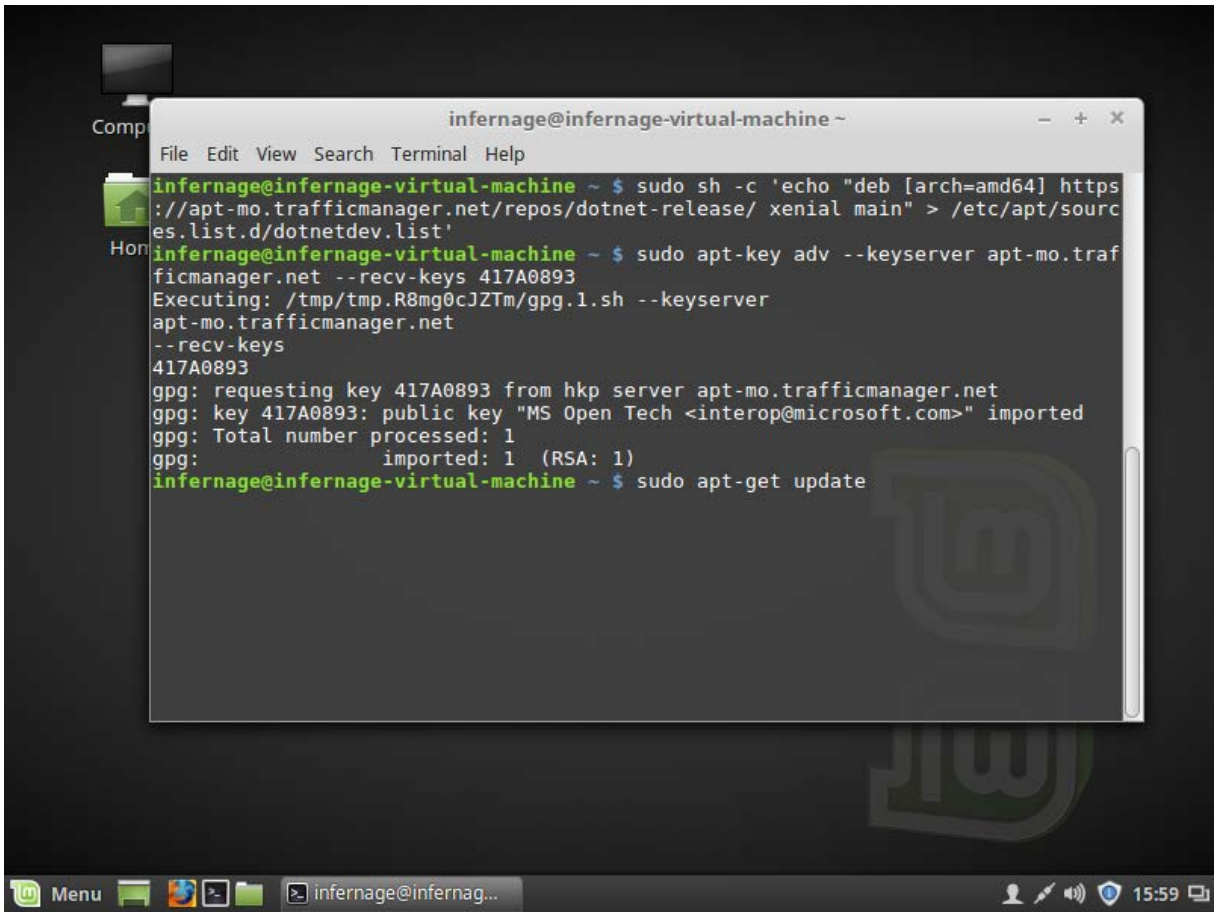

Figura 31: Actualizar fuentes de paquetes

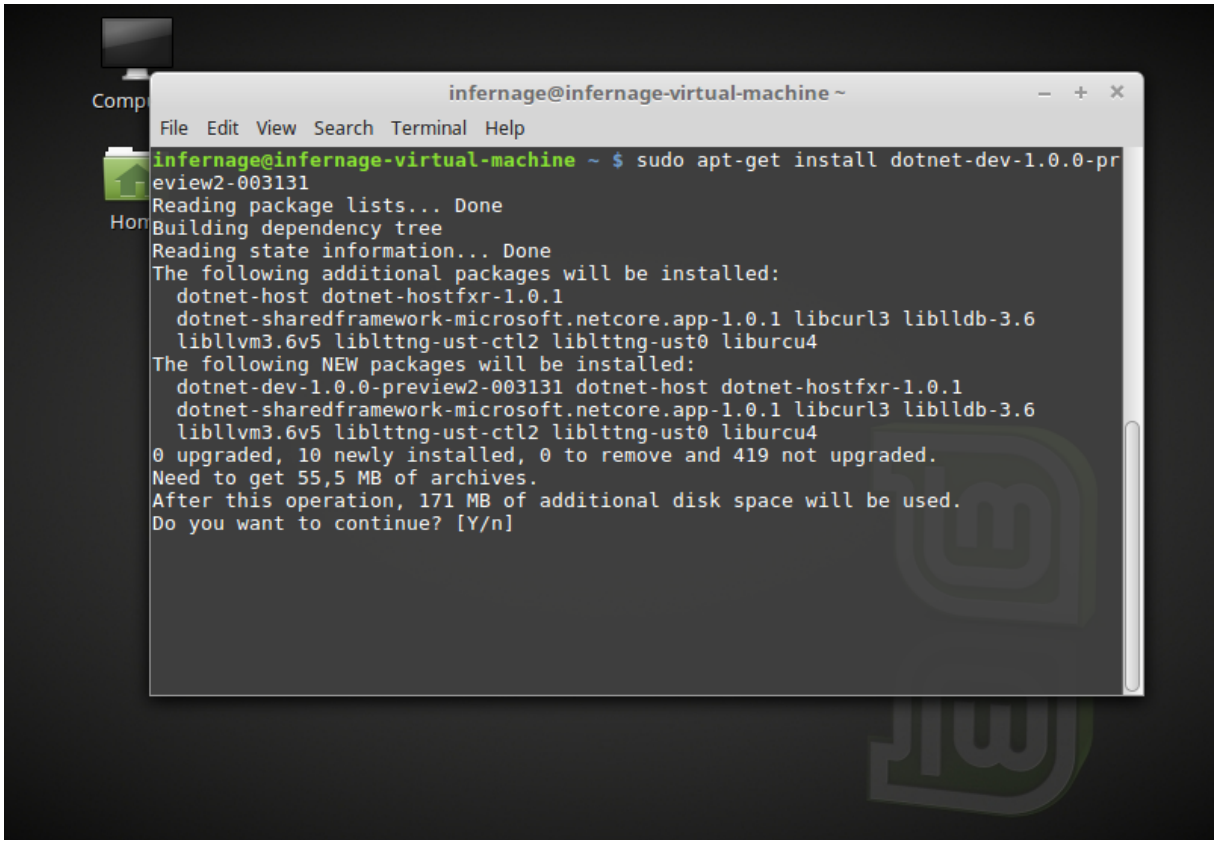

Figura 30: Instalación de .NET Core

Paso 3 - Instalación y configuración del servidor proxy Nginx

En este apartado, se instalará Nginx, además de indicar la configuración utilizada en el proyecto. Para empezar, es necesario obtener Nginx como se muestra en la Figura 32.

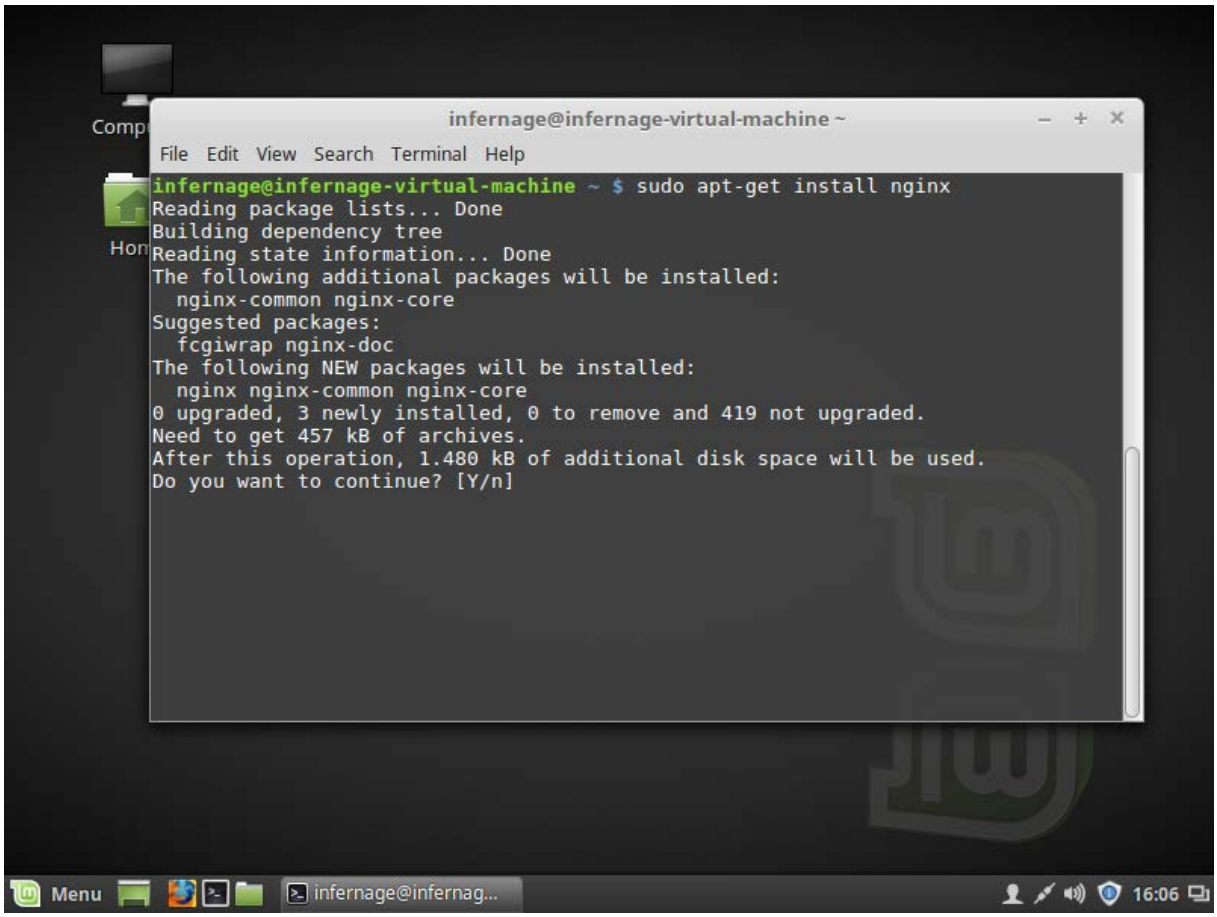

Figura 32: Instalación de Nginx

Una vez hecho esto, Nginx ya estará instalado y listo para usarse, ahora solo resta configurarlo. Primeramente, es necesario comentar o quitar el servidor de escucha que trae Nginx por defecto. Este se localiza en */etc/nginx/sites-available/default*. Para esto, es necesario añadir un # delante de cada línea que no la tenga ya, a excepción del ámbito *server*.

Con esto realizado, se procederá a añadir la configuración del servidor. Esta vez, será necesario modificar el archivo de configuración de Nginx, el cual se sitúa en */etc/nginx/nginx.conf*. Para ello, se realizarán los cambios mostrados en la Figura 33 y en la Figura 34.

```
 GNU nano 2.5.3          File: /etc/nginx/nginx.conf                Modified


        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real_IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        client_max_body_size 10m;
        client_body_buffer_size 128k;
        proxy_connect_timeout 90;
        proxy_send_timeout 90;
        proxy_read_timeout 90;
        proxy_buffers 32 4k;

        limit_req_zone $binary_remote_addr zone=one:10m rate=5r/s;
        server_tokens off;
        sendfile off;
        keepalive_timeout 29;
        client_body_timeout 10;
        client_header_timeout 10;
        send_timeout 10;
```

Figura 33: Configuración de Nginx 1

```
upstream bustrack{
        server localhost:10400;
}

server{
        listen 80 default_server;
        server_name _;
        add_header Strict-Transport-Security max-age=157680000;
        return 301 https://$host$request_uri;
}

server{
        listen *:443 ssl;
        server_name bustrack.undo.it 192.168.1.140 localhost;
        ssl_certificate /home/infernage/nginx.cert;
        ssl_certificate_key /home/infernage/nginx.key;
        ssl_protocols TLSv1.1 TLSv1.2;
        ssl_prefer_server_ciphers on;
        ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";
        ssl_ecdh_curve secp384r1;
        ssl_session_cache shared:SSL:10m;
        ssl_session_tickets off;
        ssl_stapling on;
        ssl_stapling_verify on;

        add_header Strict-Transport-Security "max-age=63072000; includeSubdomains; preload";
        add_header X-Frame-Options DENY;
        add_header X-Content-Type-Options nosniff;

        location / {
                proxy_pass http://bustrack;
                limit_req zone=one burst=10;
        }
}
```

Figura 34: Configuración de Nginx 2

Con esto realizado, es necesario notificar al proceso de Nginx de que su configuración ha sido cambiada, por lo que se escribirá en consola el comando *sudo nginx -s reload* y no tiene que salir ningún error (en este caso sale el warning de que el certificado es auto-firmado).

Para más información sobre la configuración, se puede consultar este enlace.

Paso 4 - Instalación del esquema de base de datos

Para terminar, es necesario importar el esquema de base de datos ya creado. Para ello se dispone de un archivo *dbServer.backup* el cual contiene toda la información sobre el esquema en sí. Antes de nada, hay que configurar el usuario por defecto de PostgreSQL como se realiza en la Figura 35.

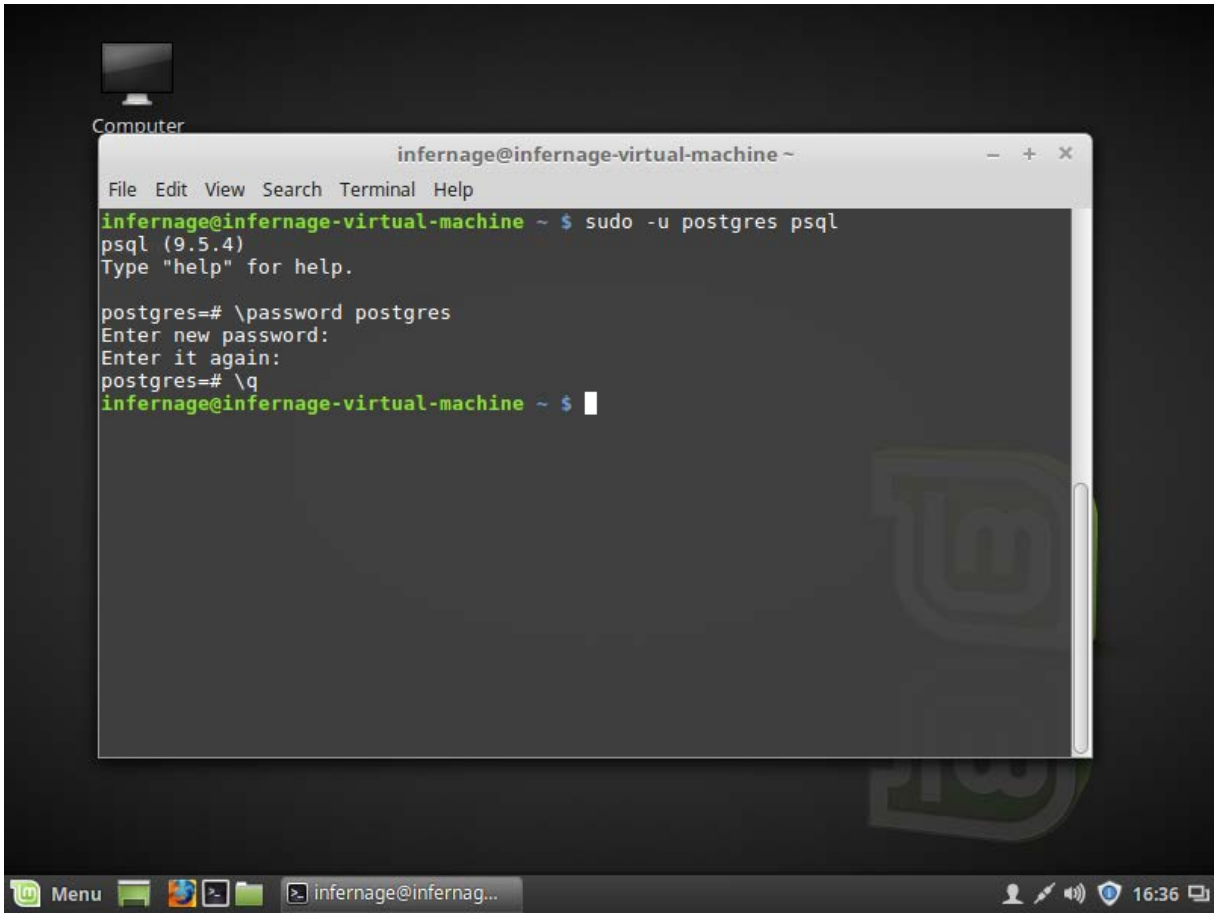

Figura 35: Cambiar contraseña de postgres

Una vez realizado el cambio, será necesario abrir la utilidad pgadmin3 (ver Figura 36) para, así, crear tanto la conexión a la base de datos (ver Figura 37) como la propia base de datos (ver Figura 38).

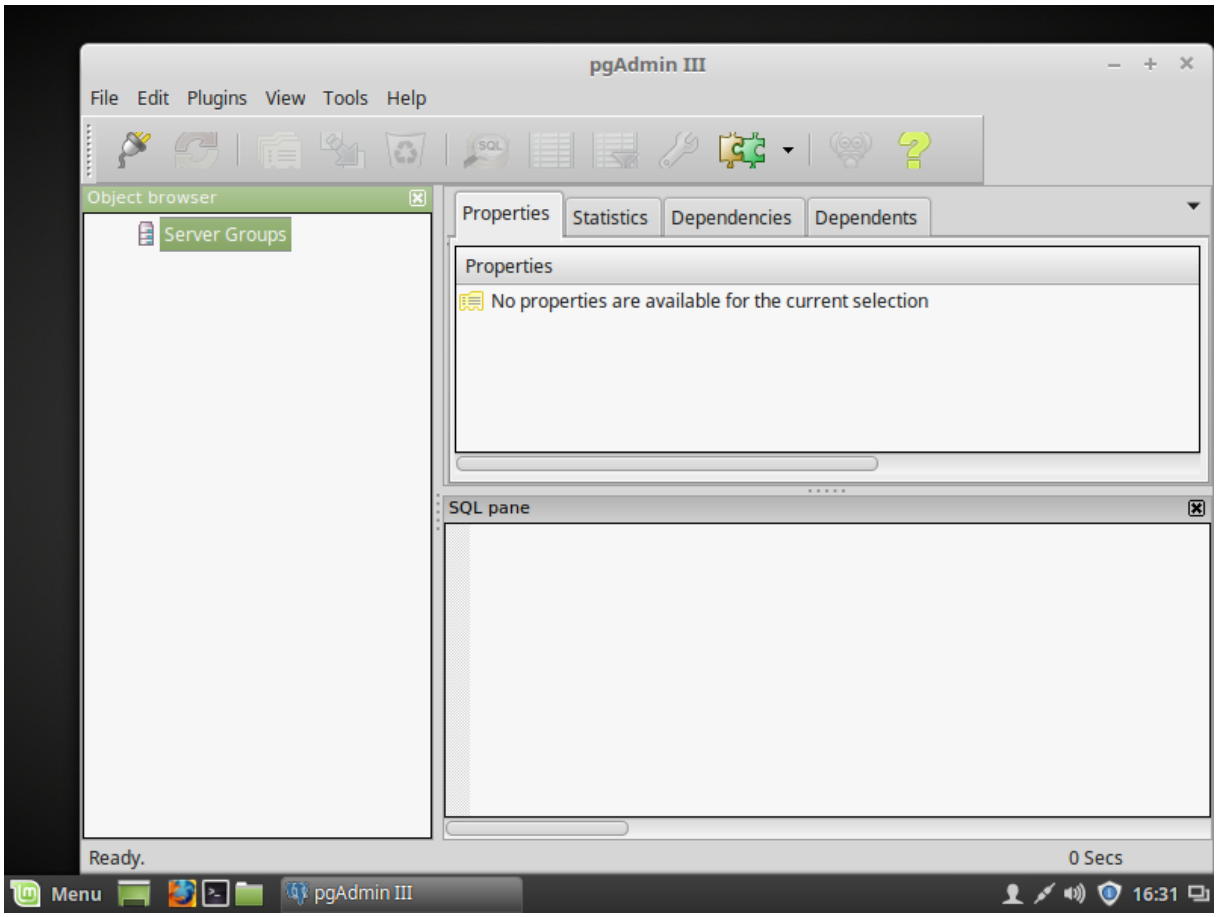

Figura 36: Interfaz de pgadmin3

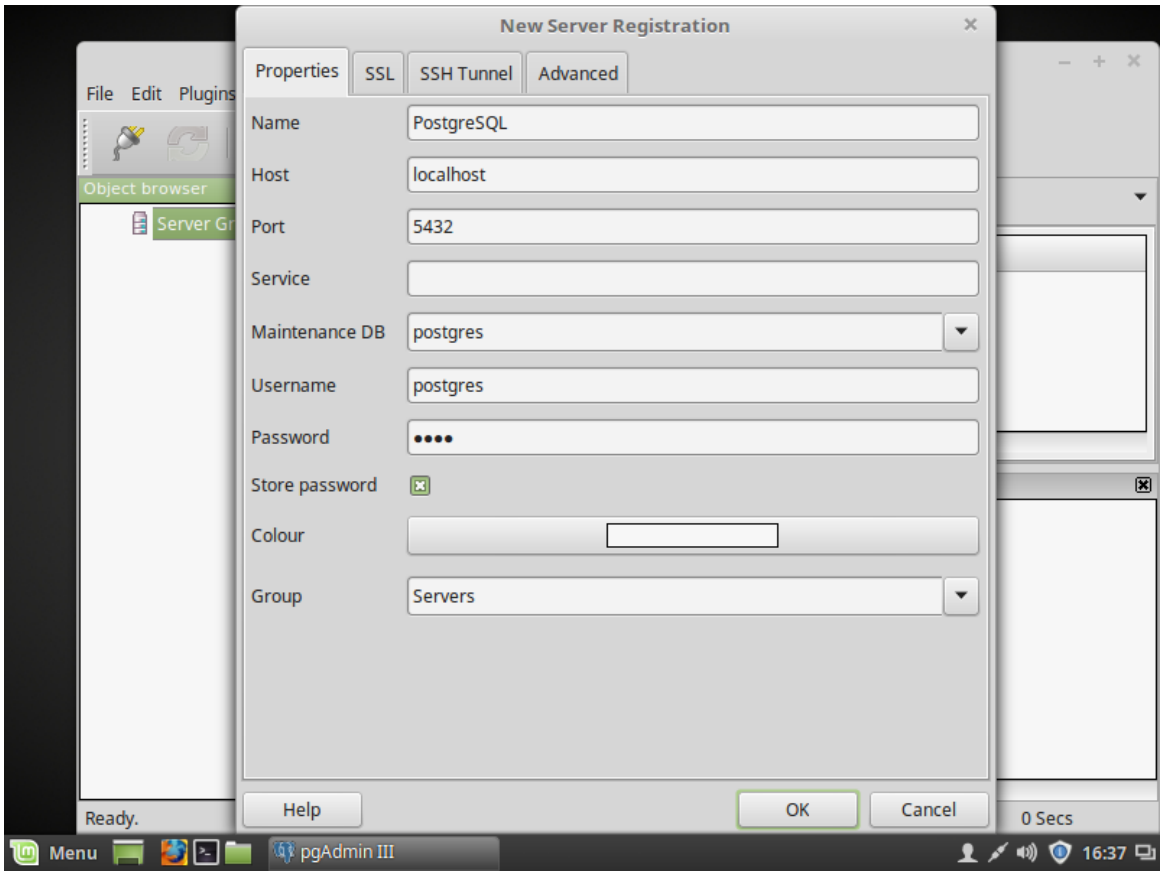

Figura 37: Creación de la conexión a la base de datos

Con la conexión creada, es necesario crear la base de datos en sí. Para ello, se abrirá la conexión y el desplegable de las bases de datos para luego hacer click derecho y seleccionar la opción de "*New database*".

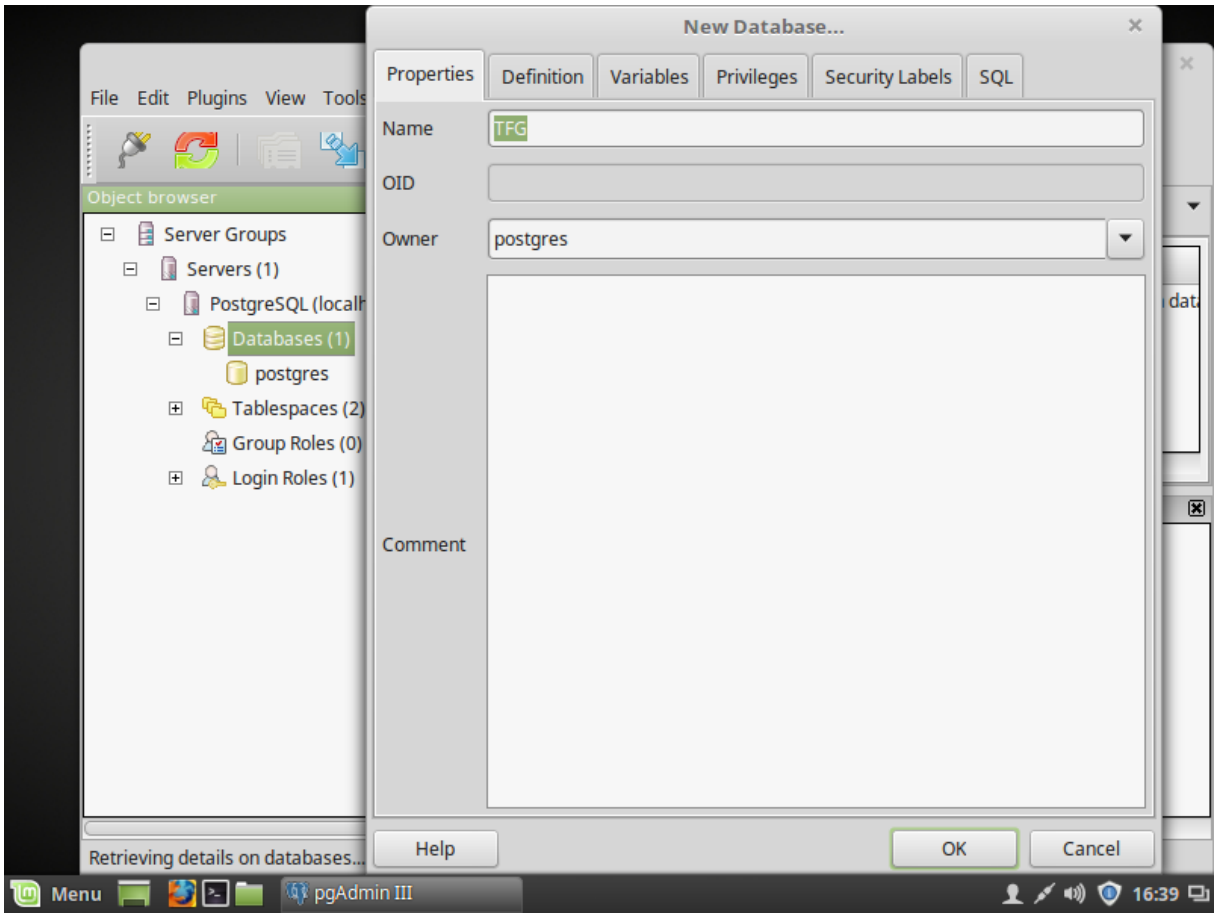

Figura 38: Creación de la base de datos

Con la base de datos creada, es hora de importar el esquema. Por tanto, es necesario abrir el desplegable de nuestra base de datos y el desplegable de "*Schemas*". En el esquema que aparece, se hará click derecho y se seleccionará la opción de "*Restore*" (ver Figura 39).

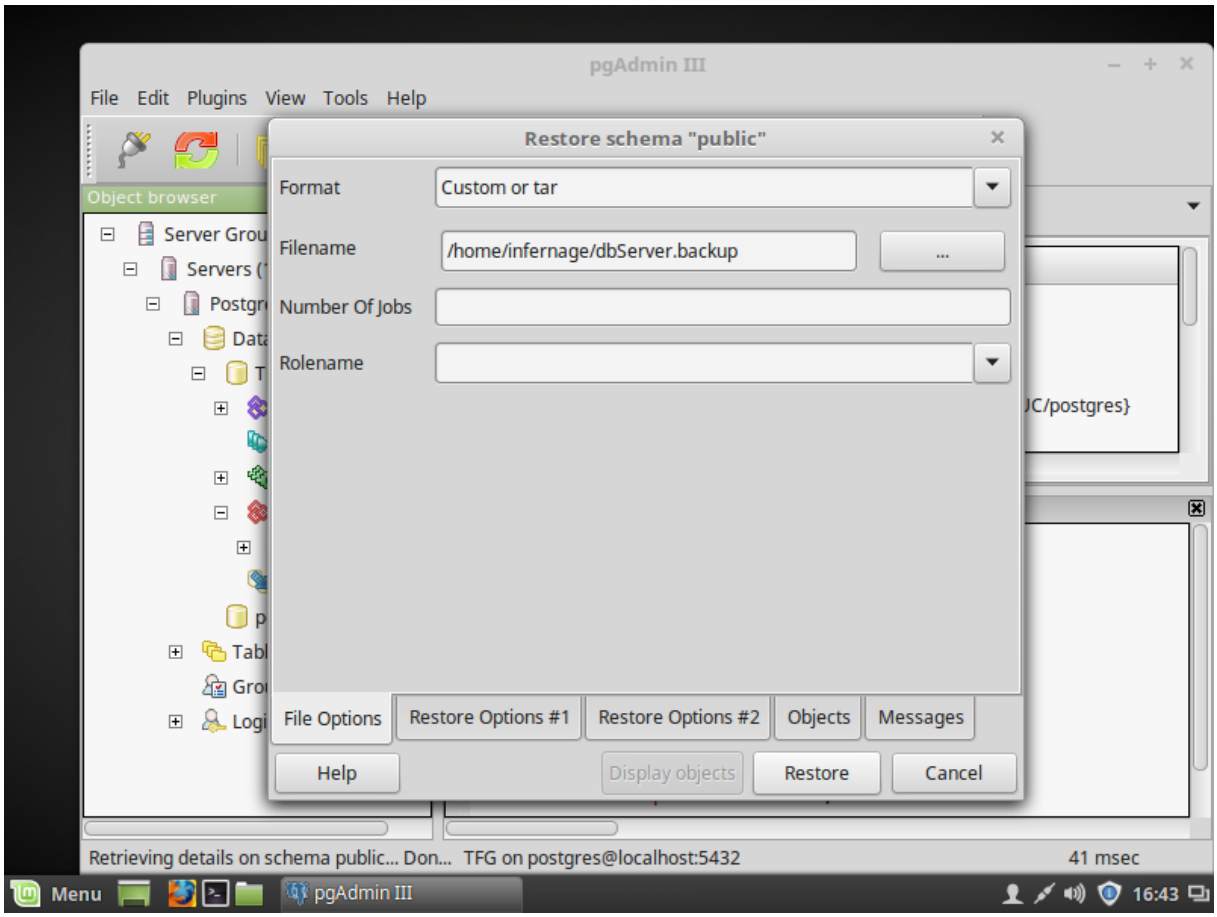

Figura 39: Importación del esquema

Una vez termine el proceso, la base de datos estará completamente operativa.

Por último, solamente queda ejecutar la aplicación con el comando de *dotnet "ruta a la dll de la aplicación"*.